Universidad Complutense de Madrid
Facultad de Informática

Proyecto de Sistemas Informáticos

Generador del Modelo Relacional y Esquemas de Bases de Datos a partir del modelo Entidad/Relación

Tutor: Miguel Ángel Blanco Rodríguez

Autores: Javier Alcolea Velázquez
Felipe Álvarez Arrieta
Lara Moreno Iglesias

Curso Académico 2006-2007

Se autoriza a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

ÍNDICE

ABSTRACT

When you create a database schema is necessary to follow some specific steps in order to get a good design.

First of all, you have to specify a conceptual representation of the mini universe the database deal with. There, the involved concepts and the relations between them will appear; this is called the ***entity-relationship diagram***.

Once you get it, you have to think about the use you are going to give to your database (e.g. which relationships will be the most looked up? Which entities will have more information?) so that the ***relational model diagram*** could be as suitable and efficient as possible.

Finally, all this information will be capture through the SQL language, which allows you to access to the database.

Our application provides the user to create databases schemas easily without knowing any SQL language but only by introducing the entity-relationship diagram and by answering some questions about the use of the database in order to get a rather efficient design.

PALABRAS CLAVE PARA BÚSQUEDA BIBLIOGRÁFICA

- Diagrama entidad-relación, diagrama E/R, diagrama entidad relación.
- Modelo relacional.
- Base de datos.
- Optimizaciones sobre un modelo relacional.
- Sentencias SQL.
- Creación del diagrama E/R.
- Generación del modelo relacional.
- Diccionario de datos.
- MySQL.

1. OBJETIVOS

En la creación de una base de datos es necesario seguir una serie de pasos para llegar a un buen diseño. En primer lugar se ha de realizar una representación conceptual sobre el mini universo que abarcará la base de datos, esto se materializa en un ***diagrama entidad-relación*** en el que aparecen los conceptos involucrados y las relaciones entre ellos. A partir de ahí y en función del uso que se le vaya a dar a la base de datos (qué relaciones van a ser más consultadas, qué entidades tienen más instancias, etc.) se escogerá la representación conceptual más adecuada para la misma. Todo ello para conseguir finalmente una representación física de nuestro mini mundo a través del lenguaje SQL.

Con este proyecto de la asignatura de Sistemas Informáticos se pretende, mediante nuestra aplicación, facilitar la creación de bases de datos partiendo del *diagrama entidad-relación* previamente pensado por el usuario. El diseño de dicho diagrama requiere un conocimiento más somero en contraposición a los conocimientos necesarios en el proceso de implementación de la base de datos en un sistema de gestión.

Sirviendo también de utilidad en el campo de la docencia, ya que se puede mostrar al alumnado la secuencia de aspectos a tener en cuenta para llegar a un buen diseño según la funcionalidad que vaya a tener la base de datos a crear (limitaciones de rendimiento, capacidad,...), así como el resultado final tras las elecciones realizadas.

Destacar que todo el proceso en los diferentes pasos de ejecución del diagrama a tratar, se presentan al usuario de forma interactiva a través de ***interfaces gráficas de usuario***; por ello, el usuario estará al tanto de las limitaciones que puedan conllevar las decisiones que toma.

El objetivo principal es llegar a un buen modelo relacional que represente el diagrama entidad-relación ideado por el usuario y mejorado gracias a los posibles itinerarios propuestos por la aplicación.

Para almacenar la información del diagrama entidad-relación pensado por el usuario se utilizará un ***diccionario de datos***, que contiene las características lógicas de los datos que se van a utilizar en el sistema que estamos programando. Se indicará si un concepto es una relación, una entidad o un atributo, así como las características interesantes en cada uno de los casos, por ejemplo, si es una relación se conocerá si es total o parcial, las entidades que participan, la cardinalidad.

El diccionario de datos permite analizar fácilmente la composición del diagrama entidad-relación que se trata en cada momento, para a partir de ese análisis advertir de posibles redundancias en la representación conceptual o proponer mejoras en la representación lógica (***optimizaciones***), como:

Por ejemplo, en el caso de una relación binaria uno a uno con participación parcial de ambas entidades, se proponen al usuario las siguientes alternativas:

- Cada vez que realice consultas respecto a esta relación le interesará sólo la información referenciada por los atributos claves de las entidades que relaciona.

- Realizará consultas y querrá obtener sobretodo información de la Entidad 1.

- Realizará consultas y querrá obtener sobretodo información de la Entidad 2.

- Estima que la Entidad 1 tendrá más información que la Entidad 2.

- Estima que la Entidad 2 tendrá más información que la Entidad 1.

Una vez completada la representación del diagrama entidad-relación y seleccionadas las preferencias para la representación lógica, es decir, para el *modelo relacional*, el **sistema generador** de la aplicación, consultando toda la información previamente recogida, creará las sentencias SQL necesarias para obtener la representación física.

2. CONOCIMIENTOS PREVIOS

2.1 ¿Para qué sirven las *bases de datos*?

Una base de datos es un conjunto de datos que pertenecen al mismo contexto almacenados sistemáticamente para su posterior uso. En este sentido, una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados (gracias al ISBN) para su consulta.

Desde el punto de vista de la Informática, la base de datos es un sistema formado por un conjunto de datos almacenados en discos que permiten el acceso directo a ellos y un conjunto de programas que manipulen ese conjunto de datos.

2.2 ¿Qué es un diagrama *entidad-relación*?

Es un diagrama conceptual gráfico que representa un mini mundo gracias a un conjunto de entidades y relaciones establecidas entre ellas que tienen sentido sobre un cierto dominio de datos.

También se puede llamar esquema entidad-relación. En muchos casos emplearemos la notación E/R para abreviar "entidad-relación".

Una **entidad** es una representación de un objeto individual concreto del mundo real. Las entidades tienen **atributos**. Un atributo de una entidad es una característica *interesante* sobre ella, es decir, representa alguna propiedad que nos interesa conocer.
Se denomina **clave** al conjunto de atributos que identifican de forma unívoca una entidad.

Las entidades se vinculan mediante **relaciones** que, en ciertas variantes de la notación, pueden también tener sus propios atributos. En principio, estas relaciones pueden ser *n-arias*, pero en la práctica se suele trabajar con relaciones binarias. Por ejemplo, una relación ternaria entre entidades A, B y C puede representarse por una nueva entidad D que tenga relaciones binarias con cada una de A, B y C. Nosotros permitiremos relaciones binarias y ternarias.

Para cada entidad pueden existir en un momento dado cero, una o muchas **instancias**. Estas instancias toman valores para sus atributos de los dominios de datos definidos para aquellos. Las instancias de una relación son pares ordenados de instancias de las entidades que dicha relación vincula.

Una relación *R* entre dos entidades **A** y **B** se puede clasificar de acuerdo con su **cardinalidad** y su **participación**:

- **Cardinalidad**
  - *R* es **uno a uno** cuando a cada instancia de **A** le corresponde una y solo una instancia de **B**.

- *R* es **uno a muchos** cuando a cada instancia de **A** le pueden corresponder varias instancias de **B**, pero cada instancia de **B** sólo se relaciona con una única instancia de **A**.
- *R* es **muchos a muchos** cuando a cada instancia de **A** le pueden corresponder varias instancias de **B** y asimismo a cada instancia de **B** le pueden corresponder varias instancias de **A**.

- **Participación**
  - *R* es **total** en **A** si para cada instancia de A existe siempre una instancia de **B** relacionada mediante **R**
  - En caso contrario, *R* es **parcial** en **A**.

2.3 ¿En qué consiste el *modelo relacional*?

El modelo relacional es la representación lógica del esquema entidad-relación. Este es el modelo de bases de datos más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente. Su idea fundamental se basa en el concepto de *tablas*, que a su vez se componen de *registros* (las filas de una tabla) y *campos* (las columnas de una tabla).

Una *tabla* es una estructura lógica que sirve para almacenar los datos de un mismo tipo (desde el punto de vista conceptual). Almacenar los datos de un mismo tipo no significa que se almacenen sólo datos numéricos, o sólo datos alfanuméricos. Desde el punto de vista conceptual esto significa que cada entidad se almacena en estructuras separadas. Así, cada entidad, tendrá una estructura (tabla) pensada y diseñada para ese tipo de entidad. Cada elemento almacenado dentro de la tabla recibe el nombre de *registro, tupla* o *fila.*

Una tabla se compone de *campos* o *columnas,* que son conjuntos de datos del mismo tipo (desde el punto de vista físico). Ahora cuando decimos "del mismo tipo" queremos decir que los datos de una columna son de todos del mismo tipo: numéricos, alfanuméricos, fechas…

En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia (a diferencia de otros modelos como el jerárquico y el de red). Esto tiene la considerable ventaja de que es más fácil de entender y de utilizar para un usuario casual de la base de datos. La información puede ser recuperada o almacenada por medio de *consultas* que ofrecen una amplia flexibilidad y poder para administrar la información.

El lenguaje más común para construir las consultas a bases de datos relacionales es SQL, *Structured Query Language* o *Lenguaje de Consultas Estructurado*.

3. DISEÑO E IMPLEMENTACIÓN

3.1 Patrones de Diseño utilizados

Para la implementación de este proyecto hemos decido utilizar un Modelo–Vista-Controlador pasivo de tres capas.

Para realizar la comunicación entre el controlador y el modelo hemos usado un patrón Fachada.

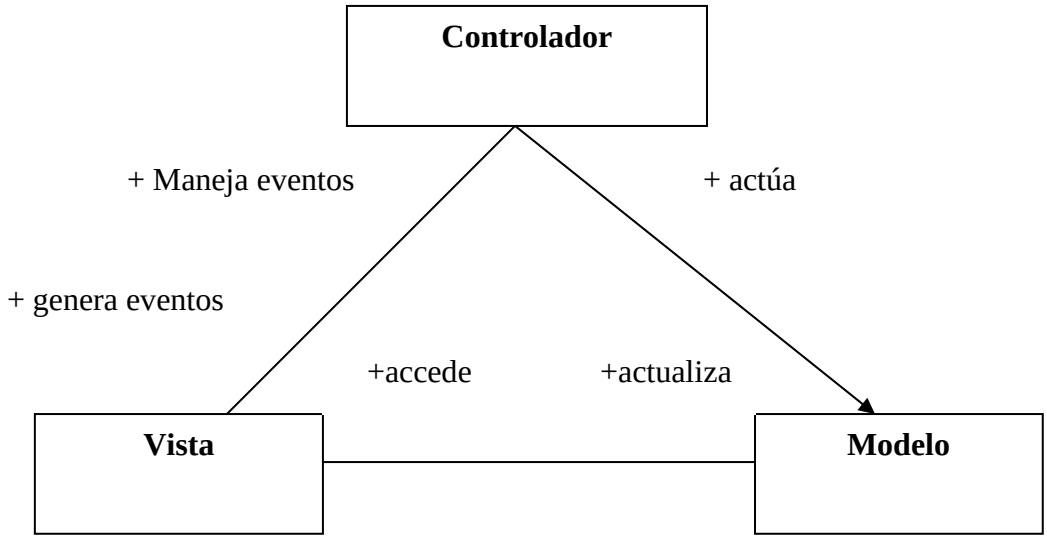
Modelo-Vista-Controlador.

El patrón/arquitectura Modelo Vista Controlador MVC divide una aplicación interactiva en tres componentes.

- El **modelo** contiene la funcionalidad básica y los datos.
- Las **vistas** muestran información al usuario.
- Los **controladores** median entre vistas y controladores.

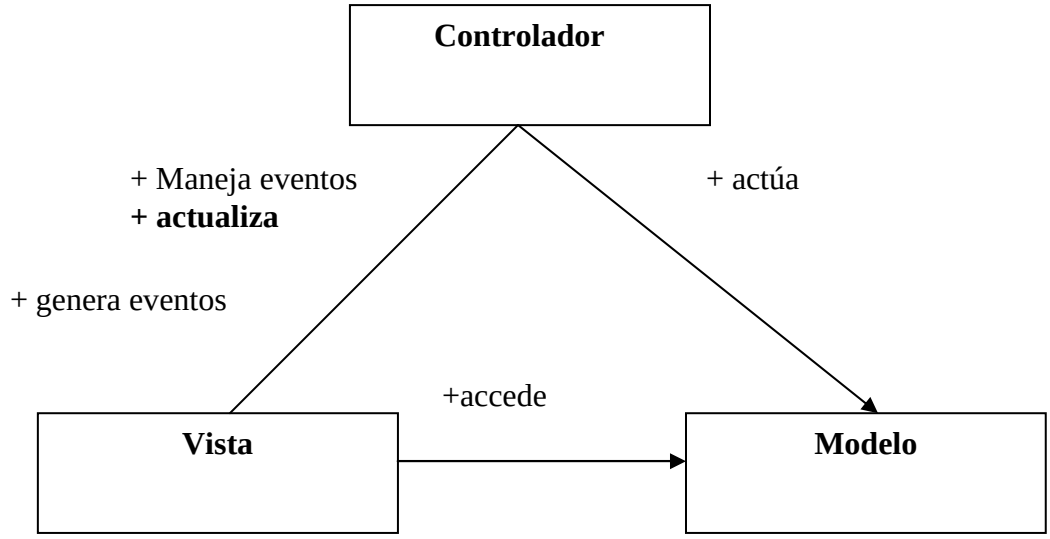
Tipos:

a) Modelo activo:
El modelo es el que actualiza la capa de vista.



La vista genera un evento, el controlador captura el evento lo trata y actúa en consecuencia sobre el modelo, el modelo actualiza la vista, y la vista accede al modelo.

b) Modelo pasivo: (El usado por nosotros en este proyecto).
Es el controlador el que actualiza la vista.



Desde la capa de vista se generan eventos y se puede acceder al modelo.

La vista genera un evento, el controlador captura el evento lo trata y actúa en consecuencia sobre el modelo, el controlador actualiza la vista, y la vista accede al modelo.

En nuestro caso la vista accede al modelo a través de un Patrón Fachada.

El controlador es el encargado de manejar los eventos del generador, actúa sobre el modelo y se encarga de lanzar los eventos de actualización de la capa de Vista.

Ventajas de este modelo:

- El modelo es independiente de la representación de la salida y del comportamiento de la entrada.

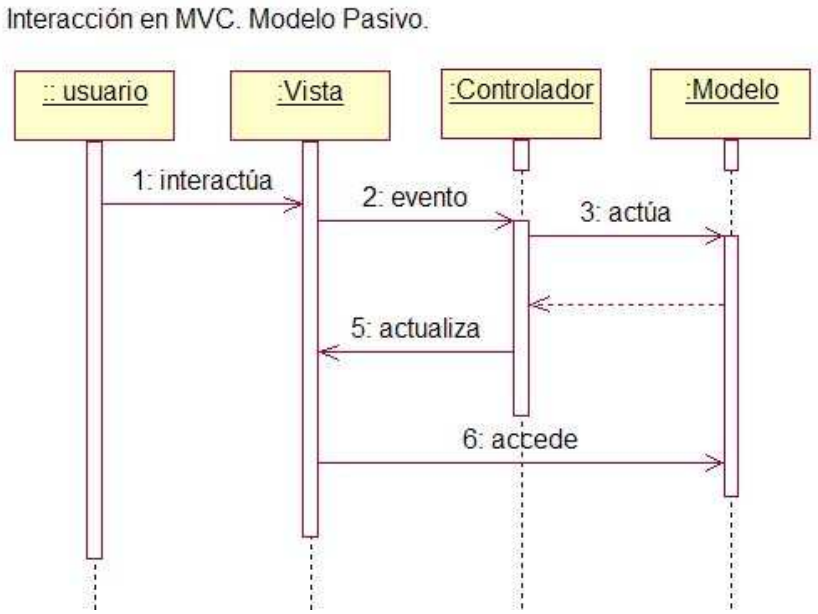
- Puede haber múltiples vistas para un mismo modelo.
- La capa de vista y la capa de modelo son totalmente independientes, lo cual es muy importante porque por ejemplo si quisiéramos realizar una versión Web futura de esta aplicación no tendríamos que modificar toda la aplicación, solamente la capa de vista, pudiendo mantener el mismo modelo (la capa de lógica y de integración).

- Los cambios son independientes en la interfaz y en la lógica.

- Se pueden realizar cambios en las interfaces sin tener que modificar la lógica, y viceversa.

- Otra ventaja, es que al tener completamente separados la Vista de la lógica, la división del trabajo del proyecto se puede realizar de manera mucho más fácil, una persona se puede ocupar de la vista, otra de la capa de lógica y otro de la capa de integración. (O por ejemplo en una empresa el departamento de desarrollo Web se puede encargar de la capa de vista, y el departamento de programación de la capa de lógica). De esta manera se reparten mejor los esfuerzos entre los miembros del equipo y se reduce de manera notable la necesidad de comunicación entre ellos.

El inconveniente que presenta es modelo es que aumenta la complejidad de su desarrollo.



Nosotros en el desarrollo de este proyecto hemos utilizado el modelo pasivo del modelo-vista-controlador.

Fachada

Hemos elegido este tipo de patrón para proporcionar una interfaz simplificada para el grupo de subsistemas de la capa de lógica.

El patrón fachada proporciona una interfaz unificada para un conjunto de interfaces de un subsistema.

Define una interfaz de alto nivel para que el subsistema de diseño sea más fácil de utilizar.

Las interfaces evitan el acoplamiento entre los subsistemas de diseño.

Los subsistemas de diseño se plasman como paquetes.

De esta manera, no solamente utilizamos interfaces, sino además una interfaz de acceso a los interfaces: la fachada.

Este patrón nos permite estructurar un sistema en subsistemas, donde cada subsistema debe implementar sus responsabilidades.

Ventajas:

- Oculta al cliente los componentes del subsistema, reduciendo así el número de objetos con los que tratan los clientes. De esta forma el subsistema es más fácil de utilizar.
- Promueve un débil acoplamiento entre el subsistema y los clientes.
- Proporciona una interfaz más fácil para el conjunto de subsistemas.
- Al introducir la fachada podemos modificar los componentes del subsistema sin afectar a los clientes.
- Además esto permite implementaciones independientes de subsistemas.

Estos son los motivos por los que en este proyecto, para acceder desde el controlador al modelo hemos utilizado una Fachada.

Modelo de tres capas.

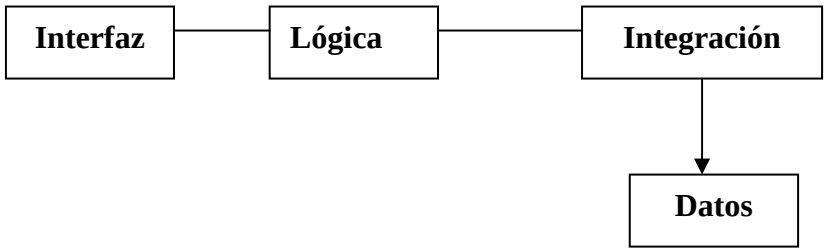
El modelo de tres capas considera una capa de presentación, otra de lógica, y otra de integración.

- La **capa de presentación** encapsula toda la lógica de presentación necesaria para dar servicio a los clientes que acceden al sistema.

- La **capa de lógica** proporciona los servicios del sistema.

- La **capa de integración** es responsable de la comunicación con recursos y sistemas externos.

Clases del subsistema:



Ventajas:

- Se puede modificar cualquier capa sin afectar a las demás.
- Facilita la división y organización o reparto del trabajo entre los miembros del equipo.
- Reduce la comunicación necesaria entre los integrantes del equipo.

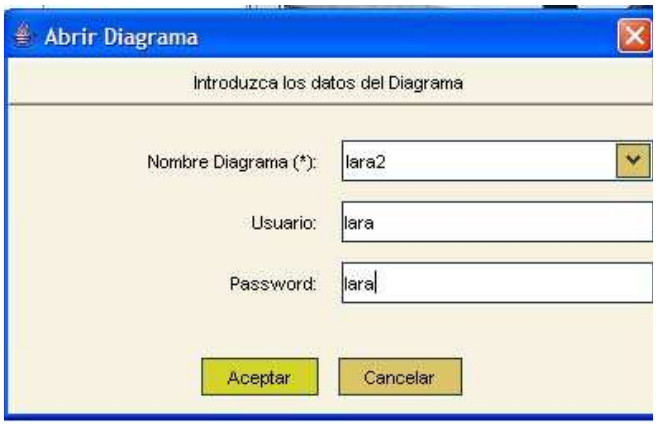
Inconvenientes:

- Mayor complejidad arquitectónica.

3.2 Ejemplo del modelo de implementación

Interfaz:

El usuario selecciona la opción Abrir diagrama.
Aparece la interfaz para Abrir un diagrama.



Al pulsar el botón **Aceptar** se lanza un evento, que es capturado por su *ActionListener*:

```
boton_aceptar.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent e)
{
boolean hay_nombre = nombre_cumplimentado();

if (hay_nombre)
{

String nombre_esquema_aux = dame_nombre_diagrama();

TEsquema taux = new
TEsquema(nombre_esquema_aux,getCampoUsuario(),getCampoPassword());

Gui_Principal.id_esquema = dame_id_diagrama(Gui_Principal.ListaEsquemas);

Controlador.getInstance().accion(EventoNegocio.DIAGRAMA_ABIERTO, taux);

       setVisible(false);
}else{
JOptionPane.showMessageDialog(null,
"Debe introducir nombre del Diagrama: campo obligatorio",
             "Error de entrada",
             JOptionPane.ERROR_MESSAGE);
}//--- fin hay_nombre
}
});//--- boton_aceptar.addActionListener
```

Desde la interfaz:

- Se recogen los datos introducidos por el usuario.
- Se chequea la validez de los datos, en este caso que se haya introducido un nombre de esquema.
- Se crea un *Transfer* del objeto esquema con los datos nombre, usuario y password introducidos por el usuario.
- Se envía al controlador el evento **EventoNegocio.DIAGRAMA_ABIERTO,** y junto con ese evento se le envía el *Transfer* del objeto esquema creado.

Controlador:

```
public void accion(int evento, Object datos)
{

switch (evento)
{

case EventoNegocio.DIAGRAMA_ABIERTO:

TEsquema esquema = (TEsquema) datos;
esquema.setIdEsquema(Gui_Principal.id_esquema);

if (fachada.validarPassword(esquema))
gui.actualizar(EventoGUI.DIAGRAMA_ABIERTO,
                     esquema.getNombreEsquema());
else
   gui.actualizar(EventoGUI.ERROR_LOGIN,
                    esquema.getNombreEsquema());
break;

}//--- fin switch
}
```

El controlador recibe los eventos, y el objeto mandado, en el caso de que el evento sea DIAGRAMA_ABIERTO, hace un *casting* del *Object* datos a un *Tesquema*.

```
TEsquema esquema = (TEsquema) datos;
```

Para acceder a la Capa de Lógica y abrir este esquema, accede a la fachada, para ello llama a la función *validarPassword(Tesquema esquema)* que es la que accede al diccionario de datos y devuelve true si el esquema existe, y el usuario y el password son correctos.

Si el resultado es *true*, lanza el evento **EventoGUI.DIAGRAMA_ABIERTO** a la interfaz principal de la aplicación, pasándole el nombre del esquema abierto:

```
gui.actualizar(EventoGUI.DIAGRAMA_ABIERTO,esquema.getNombreEsquema());
```

Si el resultado es *false*, lanza el evento **EventoGUI.ERROR_LOGIN** a la interfaz principal de la aplicación, pasándole el nombre del esquema que no se ha podido abrir.

```
gui.actualizar(EventoGUI.ERROR_LOGIN,esquema.getNombreEsquema());
```

Interfaz Principal:

Tiene un método que recibe eventos y objetos:

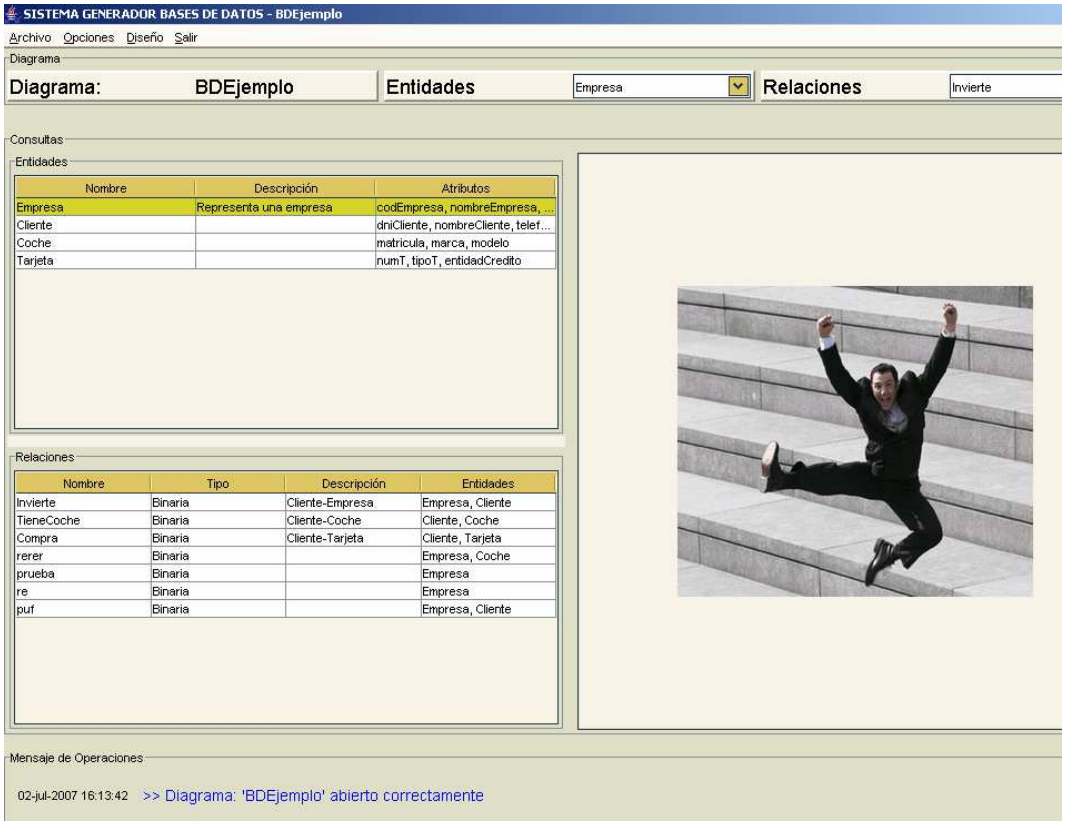
```
public void actualizar(int evento, Object datos)
{
  switch (evento)
   {
       case EventoGUI.DIAGRAMA_ABIERTO:
       {
        estilo_panel_mensajes = this.estiloAzul;// mensaje OK --> Azul
        this.panel_mensajes.setCharacterAttributes
                                  (estilo_panel_mensajes,false);
        //Sacar de datos el nombre del esquema
        nombreEsquema = (String)datos;
        this.setTitle
            ("SISTEMA GENERADOR BASES DE DATOS - " + nombreEsquema);
       this.panel_mensajes.setText
       (">> Diagrama: '"+ nombreEsquema +"' abierto correctamente");
        break;
        }
       case EventoGUI.ERROR_LOGIN:
        {
         nombreEsquema = (String) datos;
          Style estilo_panel_mensajes = null;
          estilo_panel_mensajes = this.estiloRojo;
         // mensaje error --> rojo
          this.panel_mensajes.setCharacterAttributes
                              (estilo_panel_mensajes,false);
          this.panel_mensajes.setText
    (">> Usuario y/o Password del esquema: '"+ nombreEsquema +"'
                                                        incorrectos");
          break;
          }
     }//--- fin switch

}
```

Si el evento que se recibe es **EventoGUI.DIAGRAMA_ABIERTO,** se obtiene el nombre del esquema de los datos recibidos:

```
nombreEsquema = (String) datos;
```

Y por ejemplo se actualiza el titulo de la interfaz con el nombre del mensaje, y se le muestra en los mensajes de operaciones al usuario un mensaje indicativo diciendo que el diagrama X se ha podido abrir correctamente.



Si el evento que se recibe es **EventoGUI.ERROR_LOGIN,** se obtiene el nombre del esquema de los datos recibidos:

```
nombreEsquema = (String) datos;
```

Y se le muestra al usuario un mensaje de error diciendo que el esquema X no se ha podido abrir porque el usuario ó el password eran incorrectos.

3.3 Capa Lógica

▪ *Diccionario*: En este paquete encontramos todas las clases de la lógica que interaccionan con el paquete de integración para obtener información almacenada en el diccionario de datos. Así como los tránsfers (clases de los objetos que viajan entre capas) que contienen los datos recogidos en la interfaz sobre el diagrama entidad – relación que en cada momento se esté tratando.

▪ *LogDiccionario*: Se ha usado para implementarla el patrón *Singleton*, con lo cual sólo existirá un único objeto de la misma. Esta única instancia será la que interactúe con el paquete de integración para llevar a cabo todas las consultas relacionadas con el diccionario de datos, como pueden ser: insertar nuevas entidades, ver si una relación existe ya en un esquema, obtener los atributos dada una entidad, etc.

▪ *TAtributo*: *Tránsfer* con los campos necesarios para guardar el identificador de un atributo, su nombre y su tipo (*varchar* 20, *integer, etc*)

▪ *TAtributoEoR*: Indica cuál es el identificador de un atributo que pertenece o a una entidad o bien a una relación. Al igual que las restricciones que se exigen para dicho atributo (Ej. not null) y si pertenece o no a la clave de la entidad o relación en la que se encuentra.

▪ *TAtributoRest*: Clase que hereda de atributo y que añade información de restricciones sobre el atributo e indica si es clave o no.

▪ *TConexion*: Un objeto de esta clase será donde se guarde los datos necesarios introducidos por el usuario para permitir la conexión de la aplicación a la base de datos (nombre de la base de datos, usuario y contraseña).

▪ *TEntidad*: Representa el elemento entidad de un diagrama entidad – relación. Con lo cual contiene un campo identificador para distinguir unas entidades de otras, su nombre, el esquema al que pertenece y una descripción indicando que tipo de datos guarda.

▪ *TEntidadRelacion*: Alberga el identificador de una relación y el de una de las entidades que participa en la misma. Indicando también la cardinalidad y la participación con las que interviene dicha entidad.

▪ *TEntInfoMin*: Hay momentos en los que sólo nos interesa la información mínima necesaria de una entidad (nombre e identificador). Es en este caso en el que en vez de emplear tránsfers de tipo TEntidad empelamos estos otros para evitar el desperdicio de memoria.

- *TEsquema*: En ellos guardaremos los datos necesarios que ha de introducir el usuario cuando quiere crear un esquema entidad – relación nuevo. Lo cual incluye el nombre del esquema, el usuario y la contraseña para acceder ha dicho esquema y un identificador.

- *TRelacion*: Representa el elemento relación de un diagrama entidad – relación. Contiene un campo identificador para distinguir unas relaciones de otras, su nombre, el esquema al que pertenece, una descripción indicando que tipo de datos guarda y otro atributo para saber si se trata de una relación binaria o ternaria.

- *TRelEsUn*: Tránsfer "especial" para guardar la información referente a elementos participativos de una relación es un. En concreto indicará el identificador de la entidad madre y el de uno de sus hijas, así como el identificador del esquema al que pertenecen.

- *EventoDiccionario*: Es una clase de tipos enumerados que sirven para una vez que se ha realizado algún tipo de acción en la lógica referente al diccionario de datos indicar si ésta ha concluido satisfactoriamente o no.

✓ ***Generador:*** Este paquete abarca todas aquellas clases de la lógica implicadas en la creación del modelo relacional de una diagrama entidad – relación.

- *TAtributoMR*: Representa toda la información que acompaña a un atributo en SQL cuando se incluye dentro de una entidad (relación). Es decir, el nombre del atributo, el de la entidad a la que pertenece (si pertenece a la misma entidad en la que está este campo será vacío, sin embargo, si se trata de una clave ajena irá el nombre de la entidad a la que referencia), el tipo que tiene y sus restricciones.

- *TEntidadMR*: Contiene todos los datos que se requieren al crear una nueva tabla en SQL: nombre de la entidad, lista de atributos que la componen (de la clase TAtributoMR), lista de atributos que forman parte de su clave.

- *TAtributoAnadido*: Este tipo de tránsfers se usan cuando debido a optimizaciones se le añade a una entidad un atributo que en principio no tenía. Es por ello que ha de guardar: el identificador del atributo del que se trata, el identificador de la entidad a la que inicialmente pertenecía, indicando a su vez si dicha entidad finalmente será creada o no y si pertenecerá o no a la clave de la entidad en la que se ha añadido.

- *TOptimizacion*: Guarda todas las optimizaciones a las que se ha de someter una entidad o relación (hay que indicar si se trata de una cosa o de otra mediante un campo específico), por lo que almacenará una lista de TAtributoAnadido. Puede que debido a las optimizaciones haya una entidad/ relación que no vaya a ser

creada; en este caso, también existirá una objeto de este tipo para ella indicando en el campo pertinente dicho hecho.

▪ *TRelEnt*: Tránsfer que guarda una lista con todos los identificadores de las entidades involucradas en una relación.

▪ *TRelEntMuchos*: Igual que antes (hereda de la clase TRelEnt) pero indicando además que entidades se ven afectadas por la cardinalidad "muchos" en la relación.

▪ *TrataOptimizaciones*: Clase implementada a través de un patrón Singleton que contiene una lista con todas las optimizaciones a tener en cuenta para la creación del modelo relacional del esquema en curso. A su única instancia se le irán pasando las alternativas que va eligiendo el usuario sobre la implementación del modelo relacional para que las trate y las guarde en la mencionada lista.

▪ *GuardarFichero*: Encontramos los métodos necesarios para guardar los resultados obtenidos tras el uso de la aplicación. Es decir, almacena en ficheros los modelos relacionales finales para cada esquema entidad-relación tratado. Además, también permite que las optimizaciones a aplicar escogidas por el usuario estén disponibles en cualquier otro momento en el que se recuperen los distintos esquemas manejados.

▪ *GeneraMR*: Una vez el usuario nos haya hecho saber la estructura de su diagrama entidad – relación y sus preferencias para la creación del modelo relacional, el único objeto de esta clase (volvemos a emplear el patrón Singleton) se encargará de ir consultando la lista de optimizaciones y la información del diccionario de datos que referencia a dicho esquema para agrupar la información de tal forma que se pueda crear directamente a partir de ella el modelo relacional (es decir, creará tránsfers de la clase TEntidadMR para luego mandárselos a la capa de integración y de que ésta se ocupe de generar el código SQL que sea preciso).

▪ *EventoGenerador*: Es una clase de tipos enumerados que sirven para una vez que se ha realizado algún tipo de acción en la lógica referente al generador indicar si ésta ha concluido satisfactoriamente o no.

✓ Interfaz ***IFachada***: Agrupa los prototipos de todas las funciones accesibles del paquete "lógica".

✓ Clase ***Fachada***: Implementa todos los métodos de la interfaz IFachada. Gracias a ella el Controlador podrá ordenar que se lleven a cabo las distintas acciones requeridas en cada momento sin preocuparse de que subpaquete de la lógica se encargará de ello, ya que será el objeto fachada quien distinga a quien le ha de encomendar cada tarea.

3.4 Diccionario de Datos

Toda la información referente a los esquemas entidad-relación es registrada en un ***diccionario de datos*** transparente para el usuario y común para todos los esquemas creados a través de la aplicación. La finalidad del diccionario de datos es ver posibles redundancias existentes en el esquema E/R que se esté creando y alertar de ellas al usuario, por ejemplo: Si en dos entidades diferentes se guarda un mismo atributo llamado *CódigoEmpresa*, puede que se esté guardando información dos veces y que no interese (aunque en algunos casos, a pesar de ser redundante, es lo que al usuario le conviene).

O si por ejemplo, se establece una relación entre dos entidades que ya están relacionadas, ha de avisarse al usuario y asegurarse de que quiere crear esa nueva relación a pesar de que ya haya una. O alertar si ya existe una relación en esa base de datos con ese nombre.

El diccionario de datos se ha implementado a través de una base de datos y se organiza de la siguiente manera (las claves están subrayadas y las claves ajenas en cursiva):

- Entidad (nombre, descripción, <u>códigoEnt</u>, *códigoEsquema*).
  Información sobre las diferentes entidades.
- Relación (nombre, descripción, <u>códigoRel</u>, tipo, *códigoEsquema*).
  Información sobre las distintas entidades de los distintos esquemas.
- Atributo (nombre, tipo, <u>códigoAtributo</u>).
  Guarda los datos de los atributos. Nótese que no incluye un campo restricciones, ya que de esta forma ahorramos espacio. Por ejemplo, imaginemos que tuviéramos el atributo "apellido" de tipo "varchar" y contenido tanto en la entidad "empresario" como "trabajador", si para *empresario* exigimos que tenga una longitud menor que 20 y para *trabajador* menor que 15, si en la tabla *Atributo* incluyéramos el campo *restricciones* tendríamos que crear dos filas diferentes; mientras que como lo tenemos implementado sólo se creará una fila en *Atributo* y en *AtributoEntidad* se señalarán los requisitos (lo cual implica sólo una columna más).
- AtributoEntidad (*<u>códigoAtributo</u>*,*<u>códigoEntidad</u>*, clave, restricciones).
  Relación entre una entidad y uno de sus atributos.
  Nótese que toda entidad tiene que estar relacionada con al menos un atributo.
- AtributoRelación(*<u>códigoAtributo</u>*,*<u>códigoRelación</u>*,clave,restricciones).
  Relaciona una relación con uno de sus atributos, en caso de que tenga, indicando si el mismo pertenece o no a la clave de la misma y las restricciones que se imponen sobre él.

- EntidadRelación(*<u>códigoRelación</u>*,*<u>códigoEntidad</u>*,participación, cardinalidad).
  Refleja las relaciones entre entidades, para ello cada entidad se relaciona con la entidad que la une con la otra por separado.
- RelaciónEsUn (código, *<u>códEntMadre</u>*, *<u>códEntHija</u>*, *códigoEsquema*).
  Guarda las relaciones *es un* de los distintos esquemas, indicando quienes son las entidades hijas y quienes las madres.
- Esquema (nombre, <u>códigoEsquema</u>).
  Nos indican los distintos esquemas E/R plasmados en el diccionario de datos.

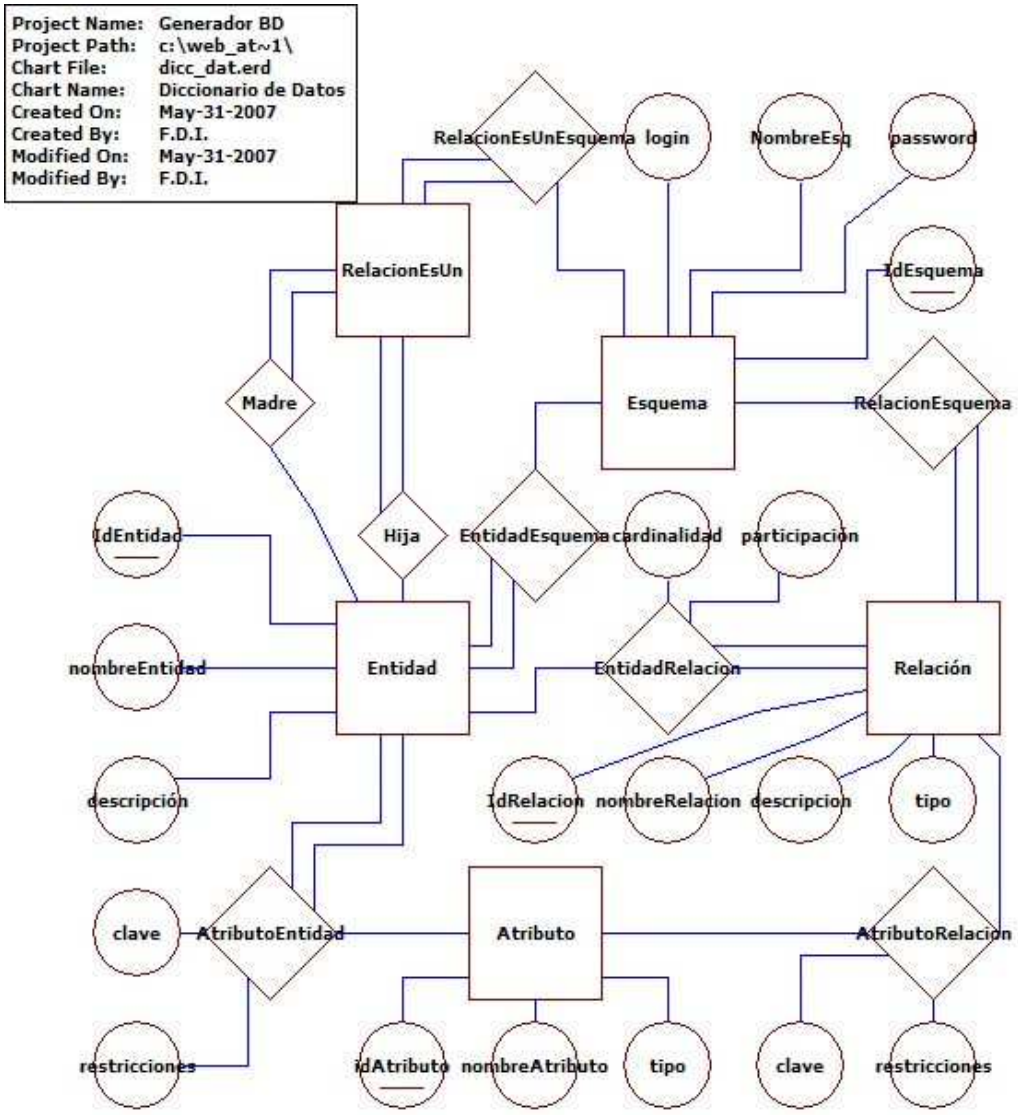
Donde:

"*códigoEsquema*" representa el esquema al que pertenece una entidad o una relación.

"*tipo*" en una relación indica si es binaria o ternaria y en un atributo si es varchar, int, Test, etc.

"*restricciones*" hace referencia a si no se permiten valores nulos (not null), si el valor del atributo tiene que estar dentro de un rango, etc.

"*clave*" indica si un atributo pertenece a la clave o no de una entidad o una relación.



3.5 Capa de Presentación

La capa de presentación es la capa de Vista de nuestro modelo de desarrollo **MODELO-VISTA-CONTROLADOR.**

La capa de presentación es la capa que permite al usuario interaccionar con la aplicación.

Hemos desarrollado la capa de presentación utilizando tecnologías java (*AWT y Swing*).

Al desarrollar la capa de presentación hemos tenido presente lo acordado en el documento de explicación informal del proyecto: "**La entrada de datos deberá ser lo más amigable e intuitiva posible**."

Es decir, lo más importante es que el diseño y programación de esta capa, le permitan al usuario utilizar las funcionalidades del sistema de una forma segura, eficiente, y amistosa.

Hemos desarrollado una aplicación monousuario, cuya finalidad es la de un proyecto académico, por lo que hemos considerado que el diseño de las interfaces gráficas debería ser lo más sencillo y claro posible. Por ejemplo, si el proyecto hubiese sido el de una página Web multiusuario en la que vendemos productos online, entonces sí que el diseño de las interfaces sería muy importante y tendría que ser bastante atractivo, llamativo, …etc, para atraer a los posibles clientes en Internet.

Como decimos al ser una aplicación monousuario académica, hemos considerado que las interfaces deberían ser lo más sencillas posibles, centrándonos en el cumplimiento de los requisitos del proyecto, y en que permitan cumplir las funcionalidades de la aplicación.

También hemos desarrollado las interfaces graficas intentando que sean lo más mantenibles posibles.

A continuación vamos a desarrollar los puntos más importantes que hemos tenido que afrontar en el desarrollo de esta capa.

3.5.1 Diseño de las Interfaces Gráficas

De manera general para el desarrollo de cada interfaz hemos partido de un boceto, un simple borrador dibujado en papel de cómo debería ser dicha interfaz.

Para realizar dicho boceto, nos hemos reunido los integrantes del equipo, y entre todos hemos revisado, que datos de entrada, datos de salida, funcionalidades… debía tener cada interfaz.

Por ejemplo: En el caso de la interfaz gráfica de "Abrir diagrama" acordamos que la interfaz debería tener 3 campos (diagrama, usuario y password), que en el campo diagrama se deberían mostrar todos los diagramas existentes, permitiéndole al usuario seleccionar un diagrama. Una vez seleccionado el diagrama, introducido su usuario y su password, al pulsar sobre el botón de aceptar, el sistema debería abrir el diagrama, y mostrar un mensaje al usuario informándole de si el diagrama ha sido abierto, ó en el caso de que no se halla podido abrir, un mensaje indicando las causas del error.

Una vez que tenido un boceto de la interfaz gráfica acordado, hemos procedido a su implementación, hemos introducido la interfaz en la aplicación y hemos comprobado si cumplía con los requisitos y funcionalidades requeridos. También hemos discutido en reuniones entre nosotros como podíamos mejorar dichas interfaces, o como podíamos simplificar los procesos, por ejemplo en una versión inicial para dar de alta un entidad teníamos que pasar por 3 interfaces gráficas, una para definir la entidad (Editor de Entidades), en otra interfaz definíamos los atributos (Editor de atributos), y en otra las restricciones de dichos atributos en la entidad especifica, después de muchas mejoras hemos llegado hasta la versión 4 del editor de entidades que realiza todo este proceso en un único formulario.

Cabe destacar que fruto de dicho trabajo de desarrollo y comunicación entre los integrantes del equipo se han logrado unas mejores interfaces gráficas, y procesos de introducción de datos más sencillos.

1.1) Interfaz Principal:

En la interfaz gráfica principal de la aplicación distinguimos los siguientes apartados:

Esquema inicial:

A) Barra Menú de Operaciones
B) Seleccionador de Entidades y Relaciones. Mostrara el nombre del diagrama, y las entidades y relaciones de dicho diagrama.
C) **Zona de Consultas:** Se podrá visualizar información de las entidades y las relaciones generadas utilizando la aplicación.
D)**Mensaje de Operaciones**: Aquí aparecerán mensajes explicativos con el resultado de las operaciones. Ejemplo: "***Entidad creada correctamente***"

A) **Barra de Menú de Operaciones**:

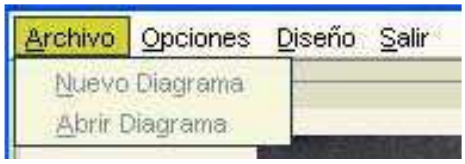
Tanto el menú, como todas sus opciones tienen teclas de acceso para mejorar su accesibilidad y usabilidad.

A.1) Archivo.

A.1.1) Nuevo Diagrama. → Permite al usuario crear un nuevo diagrama.

A.1.2) Abrir Diagrama. → Permite al usuario abrir un diagrama ya existente.

Ambos botones inicialmente estarán inactivos hasta que no se establezca la conexión con la Base de Datos que utiliza la aplicación.



A.2) Opciones.

A.2.1) Conexiones BD. → Permite la configuración de la conexión a la base de datos.

A.2.2) Generar Tablas. → Permite la generación del Modelo Relacional.

A.2.3) Generar SQL. → Permite la generación del código SQL.

La opción Conexiones BD estará inicialmente activa, la opción Generar tablas no se activará hasta que exista por lo menos una entidad en un diagrama, la opción Generar SQL no se activará hasta que no se halla generado previamente el Modelo Relacional.

A.3) Diseño.

A.3.1) Entidad. → Dar de alta entidades.

A.3.2) Relación. → Dar de alta relaciones.

A.3.3) Consultas. → *Funcionalidad disponible en la próxima versión*.

Las opciones de Entidad, Relación y diseño no estarán activas hasta que no se halla creado un nuevo diagrama ó no se halla abierto un diagrama existente previamente.

A.4) Salir.

Cerrar → Salir de la aplicación.

Esta opción permanecerá siempre activa.

B) Seleccionador de Entidades y Relaciones:

Esta sección no será visible hasta que no se halla abierto un diagrama, o se halla creado uno nuevo.

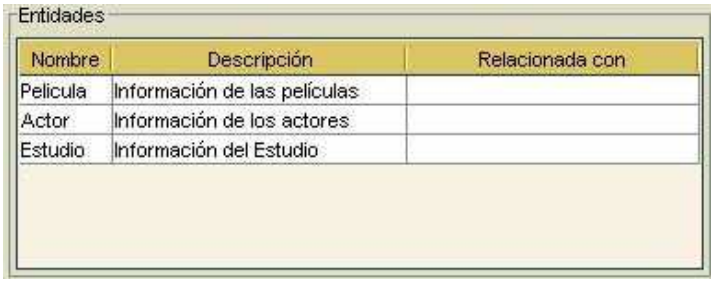
Muestra el nombre del diagrama actual, sus entidades y relaciones.

C) Zona de Consultas:

No se mostrará hasta que no se halla abierto un diagrama, o se halla creado uno nuevo.

C.1) Información de entidades:

Proporciona información de las entidades del Diagrama, su nombre, descripción y entidades con las que esta relacionada.



C.2) Información de relaciones:

Proporciona información de las relaciones del Diagrama, el nombre de la relación, su tipo (Binaria, Ternaria), y su descripción.

D) Mensaje de Operaciones:

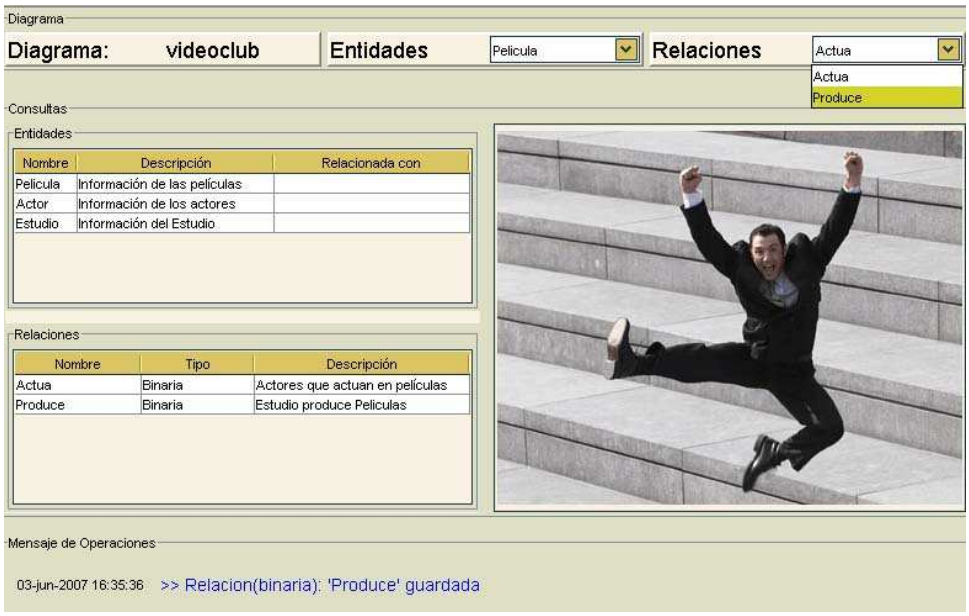
A través de esta zona mostraremos mensajes explicativos que informaran al usuario del resultado de las operaciones realizadas con la aplicación

Por ejemplo:

Si el usuario se ha conectado correctamente a la base de datos, mostrará en azul.

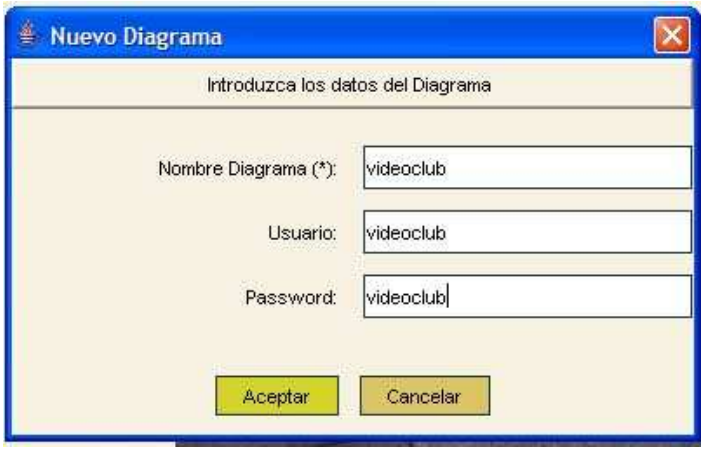
Si la hora de querer abrir un diagrama existente, la operación no se puede realizar, se le muestra un mensaje explicativo en rojo, sobre la causa se dicho fallo.

Ejemplo de Interfaz Principal con un diagrama abierto:



1.1) Interfaz Nuevo Diagrama:

Permite al usuario crear un nuevo diagrama.



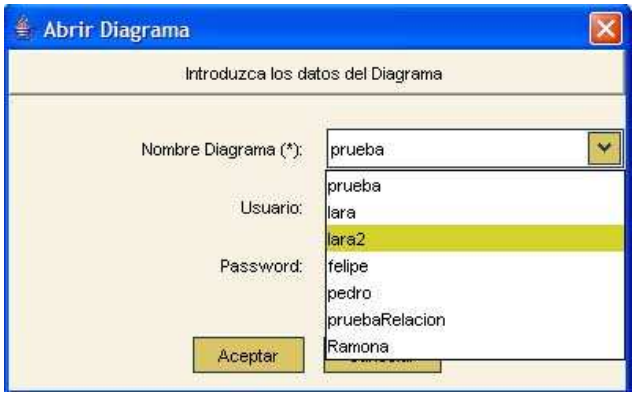
El usuario debe introducir el nombre que le quiere dar al nuevo diagrama.

El nombre del programa es un campo obligatorio, hasta que no se cumplimente no se podrá crear un nuevo diagrama.

Puede crear un usuario y una contraseña para dicho diagrama, introduciendo sus valores en dicho formulario.

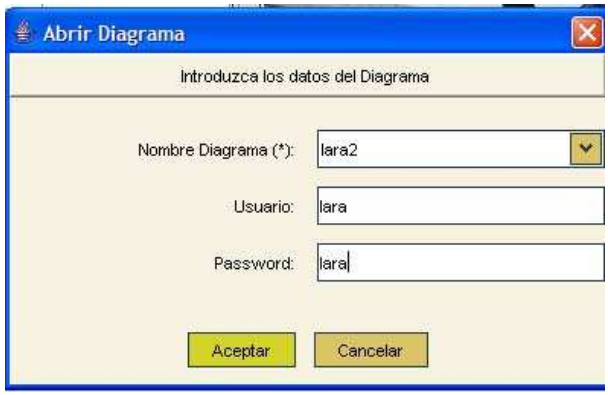
1.2) Interfaz Abrir Diagrama:

Permite al usuario abrir un diagrama ya existente.



El usuario puede seleccionar el diagrama que desea abrir de los existentes en la base de Datos.

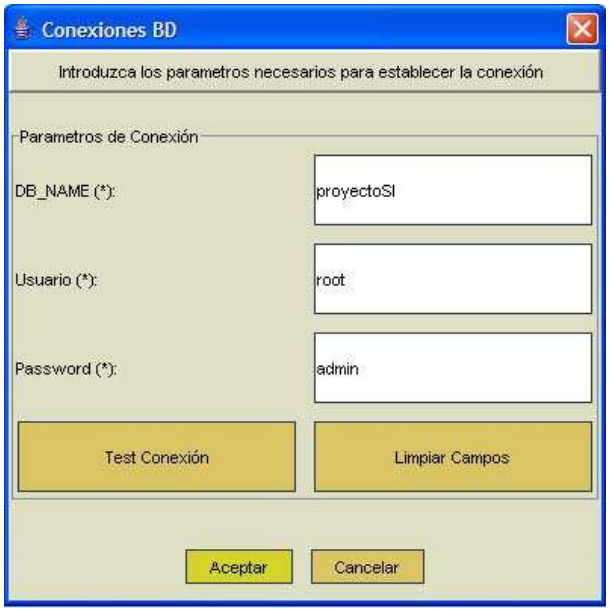
Una vez seleccionado un diagrama, podrá introducir el usuario y la contraseña de dicho diagrama (en el caso que dicho diagrama tenga un usuario y una contraseña definidos).



Al pulsar en aceptar si el usuario y la contraseña coinciden con el del diagrama seleccionado, este pasará a ser el diagrama utilizado por la aplicación.

1.3) Interfaz Conexión BD:

Permite la configuración de la conexión a la base de datos.



El usuario deberá introducir el nombre de la base de datos, el usuario y el password.

Estos tres campos son obligatorios, hasta que no se cumplimenten no se procederá a intentar la conexión a la Base de datos.

Al pulsar el botón "*Limpiar Campos*" se eliminan los valores introducidos en *DB_NAME*, Usuario, Password.

El pulsar el botón "Test Conexión" se comprueba si ya existe una conexión previa a una Base de Datos.

1.4) Interfaz Alta Entidad:

Permite al usuario dar de alta Entidades en el sistema.

Esta interfaz consta de las siguientes zonas.

Nombre de la Entidad:

Descripción de la Entidad:

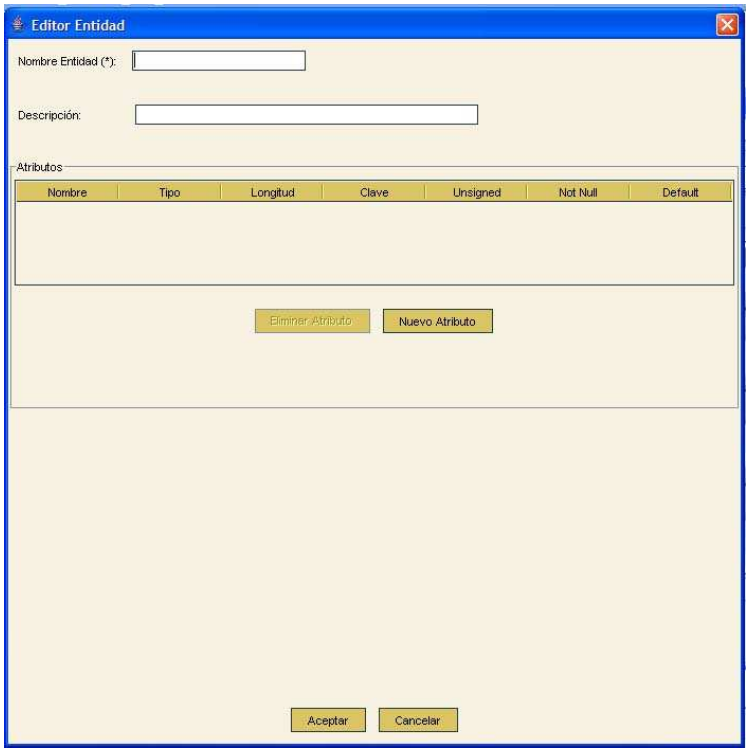
Visualizador de los atributos de la entidad:

Nombre	**Tipo**	**Longitud (Longitud/No)**	**Clave (Si/No)**	**Unsigned (Si/No)**	**Not Null (Si/No)**	**Default (valor)**
Nombre del atributo	*int, Varchar, date,...*	Si tiene atributo → Longitud. Si no → No	Si el atributo es clave o no	Si el atributo el *unsigned* o no.	Si es requerido	Su valor por defecto

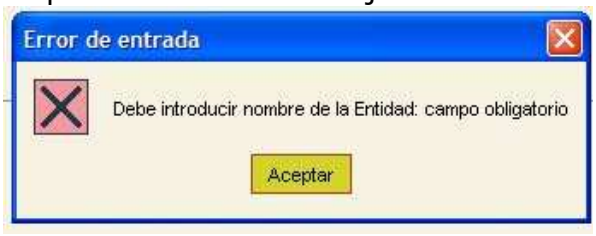
Muestra los atributos que tiene la entidad, su tipo, si es clave o no, y sus restricciones.

Editor de Atributos:

El usuario puede definir el nombre de la entidad y su descripción, introduciendo sus valores en los campos respectivos.



El campo Nombre de la entidad es obligatorio, hasta que no se rellene no se podrá dar de alta una Entidad, si se pulsa aceptar y no hay nombre definido aparecerá este mensaje:



Al pulsar el botón "Añadir Atributo" aparece un panel de Edición de Atributos, que le permitirá al usuario definir los atributos que tiene la entidad.

En este editor de atributos el Usuario puede definir el nombre del Atributo, y su tipo. En función del tipo podrá definir unas características adicionales. Por ejemplo si el tipo en **int** o **Varchar** podrá definir su Longitud, pero el cuadro de edición longitud no se mostrará si el tipo es **Date**.

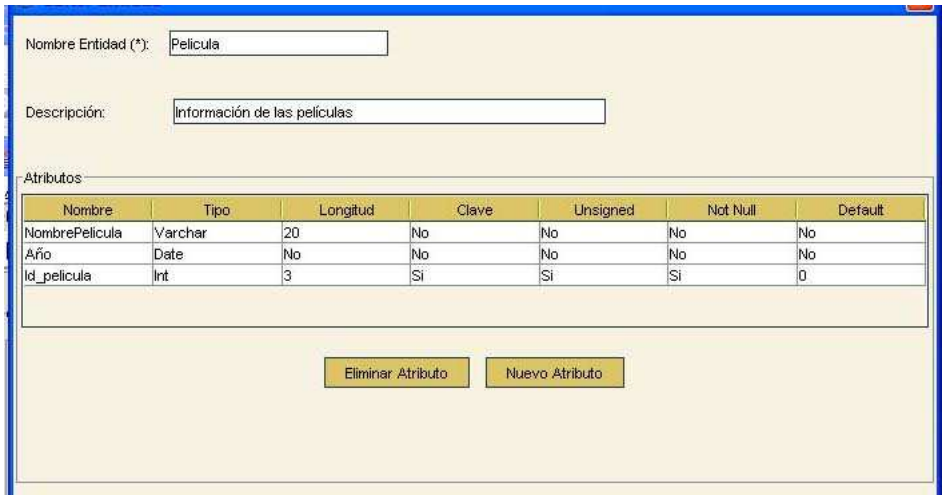
Es decir el Editor de atributos es dinámico y cambia en función del tipo de datos seleccionado, para cada tipo mostrará los campos correspondientes que se pueden definir.

Además el usuario podrá definir si dicho atributo es **clave** de la relación, y asignar a dicho atributo una serie de restricciones (como hemos dicho en función del tipo). Por ejemplo si el tipo en **Int,** podremos definir si es clave, si es requerido, un valor por defecto y su longitud.

El campo Nombre Atributo es obligatorio, es decir hasta que no se defina no se podrá añadir el atributo, si lo intentamos y no hay nombre de atributo definido se le ofrecerá este mensaje de información al usuario.

Si pulsamos el botón "Limpiar campos" se limpian todos los valores del editor de atributos.

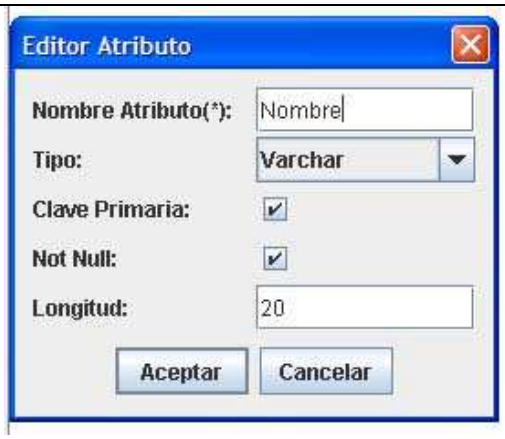
Si pulsamos el botón "Añadir Atributo", si se ha definido su nombre, el atributo se añadirá a la entidad y se mostrará en la tabla de los atributos de la entidad.



Para poder dar de alta una entidad se deberá haber definido al menos un atributo como clave. Si no se ha definido ningún atributo como clave se le muestra al usuario este mensaje.

Si se ha definido el nombre de la Entidad, y al menos se le ha definido un atributo como clave, al pulsar el botón "Aceptar" la entidad será dada de alta en el diagrama.

A modo de ver la evolución del prototipo inicial al actual se incluye el aspecto del la versión inicial del Editor de Entidades, y el editor de atributos.

Versión 0 Editor Entidad	Versión 0 editor atributos
	

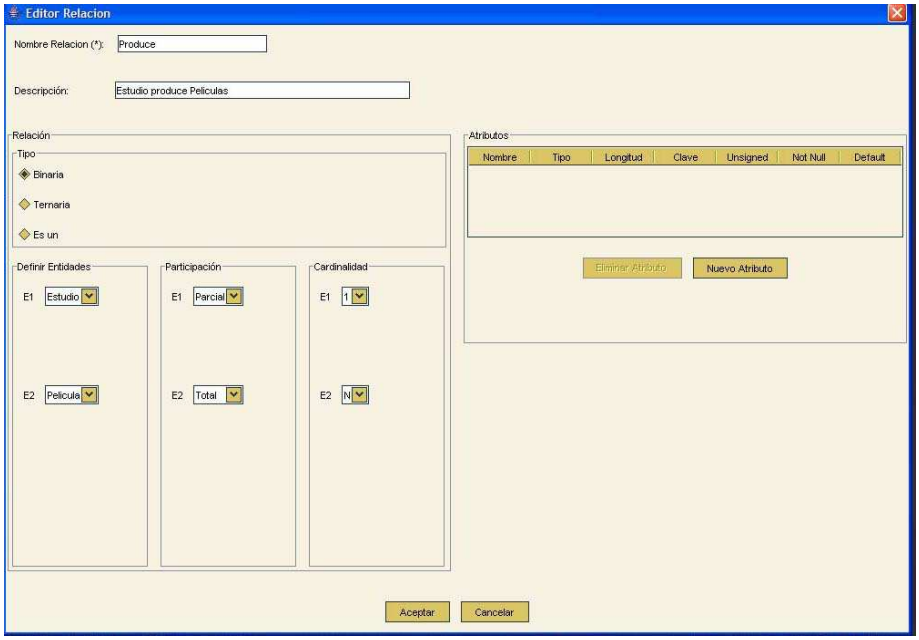
Al presionar el botón "Añadir Atributo" aparecería una nueva interfaz "Editor Atributo" en la que podíamos añadir los atributos de la entidad.

1.5) Interfaz Alta Relación:

Permite al usuario dar de alta relaciones en el diagrama que este utilizando en la aplicación.

En el editor de Relación se distinguen las siguientes zonas:

Nombre(*):			
Descripción:			
Tipo: - Binaria - Ternaria - Es un			**Tabla Atributos de la Relación**
Entidades	**Participaciones** (Total, Parcial)	**Cardinalidades** (1,N,1..N,N..M)	
Entidad 1	Participación1	Cardinalidad 1	
Entidad 2	Participación 2	Cardinalidad 2	**Editor de Atributos**
Entidad 3	Participación 3	Cardinalidad 3	



El usuario puede introducir el nombre de la relación, y su descripción en sus respectivos campos.

El nombre de la relación es un campo obligatorio, hasta que se defina no se podrá dar de alta una relación.

El usuario puede definir relaciones "binarias", "ternarias" ó relaciones "es un".

Este formulario es dinámico en el sentido que su apariencia cambia en función del tipo de relación elegido.
(De esta forma tenemos una sola interfaz para el alta de relaciones, en vez de tener una interfaz para cada tipo de relación).

1.5.1) Relación Es un:

Si se selecciona el tipo **"Es un"**, solo se muestra un campo para seleccionar el nombre de la entidad madre entre todas las entidades existentes en el diagrama, y otro campo para seleccionar el nombre de la entidad hija entre todas las entidades existentes en el diagrama.

No se podrá introducir nombre de la relación, ni descripción, ni añadir atributos (estos campos se ocultan).

Al pulsar en Aceptar se dará de alta la relación "Es un" en el diagrama.

1.5.2) Relación Binaria:

Se podrá definir el nombre y la descripción de la relación.

El campo nombre es obligatorio.

Se podrá seleccionar las entidades, la participación y la cardinalidad de las dos entidades que formen parte de la relación.

Las entidades se seleccionan de las entidades existentes previamente en el diagrama. La participación podrá definirse como Total ó Parcial, y la cardinalidad como "1" ó "N" para cada una de las entidades.

Se pueden añadir atributos en la relación, pulsando el botón añadir atributos aparece el mismo editor de atributos utilizado en la interfaz de alta de entidades. Al pulsar en el botón Aceptar del editor de atributos, el atributo será añadido a la relación y se podrá visualizar en la tabla de atributos de la relación (esta tabla de atributos es la misma que la utilizada en el formulario de alta entidad).

Al pulsar en aceptar aparecerá el formulario de Optimizaciones que en función de la participación y cardinalidad de las entidades implicadas en la relación nos mostrará unas preguntas (opciones) sobre posibles optimizaciones a realizar con dicha relación.

En función del tipo de consultas que quiera realizar el usuario sobre esta relación, y en función también del tamaño de las entidades se realizara un tipo de optimización diferente.

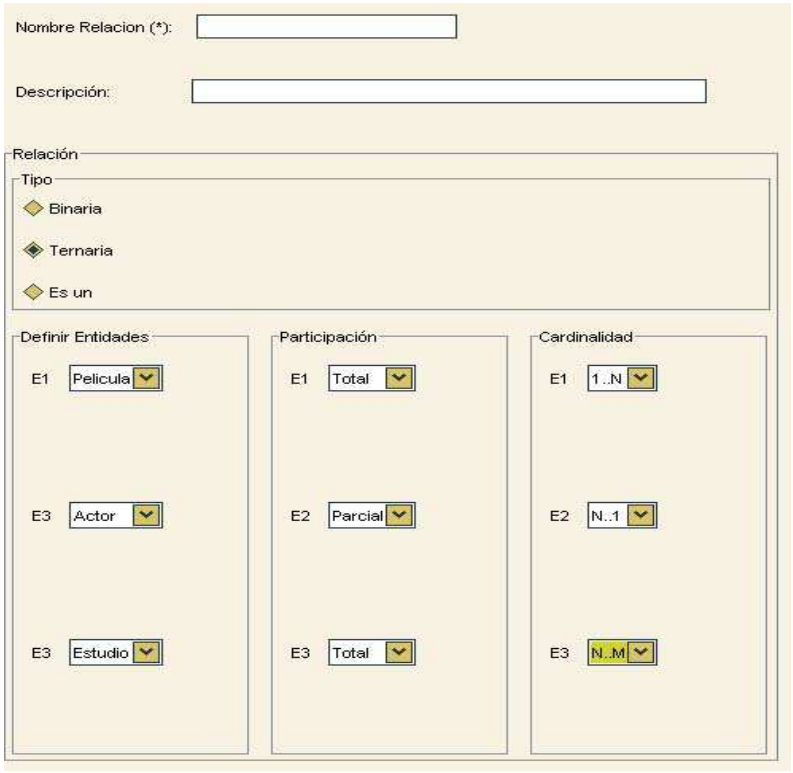
Finalmente la relación será dada de alta en el diagrama actual.

1.5.3) Relación Ternaria:

Similar a el caso de relación Binaria. En este caso se podrá seleccionar una tercera entidad entre las existentes en el diagrama, y la participación y cardinalidad de esta tercera entidad.

Se pueden añadir atributos a la relación.

Al pulsar el botón aceptar, si la relación tiene atributos se da de alta.
Si la relación no tiene atributos se le pregunta al usuario sobre la posible conveniencia de transformar la relación ternaria en dos relaciones binarias como posible optimización.



1.6) Interfaz Optimizaciones:

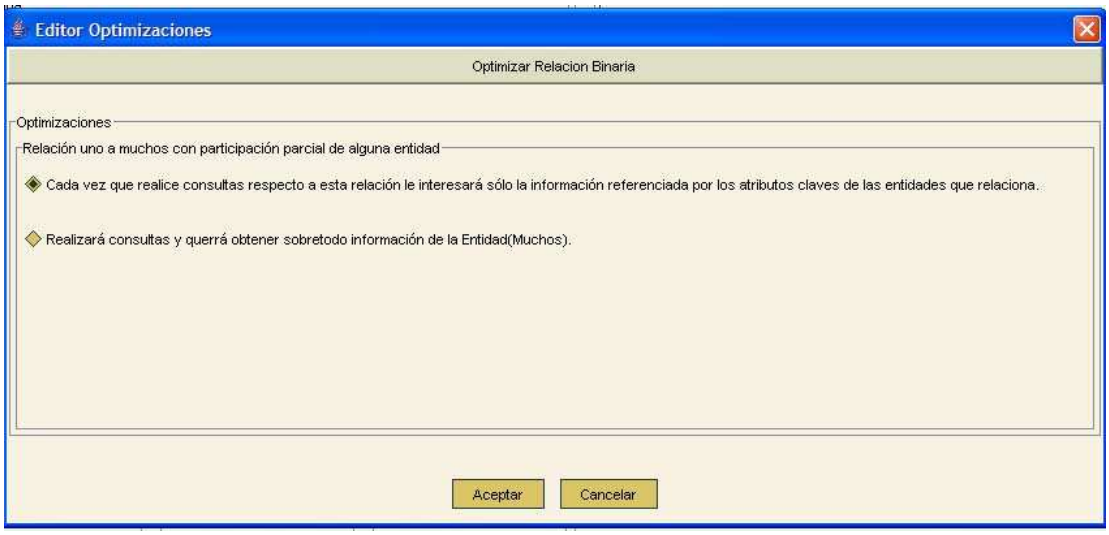
Al pulsar botón **aceptar** del formulario alta de relaciones se lanza esta interfaz.

Esta interfaz es dinámica, y en función del tipo de Relación (Binaria, o Ternaria), y la cardinalidad y participación de la entidades que forman parte de la relación, ofrece al usuario una serie de preguntas (opciones) en función de las cuales a la relación se la aplicará un tipo de optimización.

De esta manera tenemos un único formulario de optimizaciones, en vez de 1 formulario para elegir el tipo de relación y luego un formulario para cada caso de posibles optimizaciones (Relaciones 1 a 1, Relaciones 1 a N, Relaciones N a M).

En función del tipo de consultas que quiera realizar el usuario sobre esta relación, y en función también del tamaño de las entidades se aplicara un tipo de optimización diferente.

Ejemplo: Relación uno a uno con participación parcial de alguna entidad.



3.5.2 Estructura de paquetes de las interfaces gráficas

- ✓ **principal** → *Principal.java*.

- ✓ **controlador** → paquete del controlador.

- ✓ **presentacion**
  - *EventoGui.java*
  - *Gui_Principal.java*
  - *IGui.java*

- ✓ **presentacion.archivo:**
  - *GuiAltaDiagrama.java*
  - *GuiAbrirDiagrama.java*

- ✓ **presentacion.entidades**
  - *Gui_Alta_Entidad_v3.java.*
  - *ModeloTablaAtributos.java*
  - *ModeloTablaEntidades.java*

- ✓ **presentacion.imagenes**
  Imágenes utilizadas en la aplicación.

- ✓ **presentacion.opciones**
  - *Gui_conexiones_BD.java*

- ✓ **presentacion.relaciones**

  - *Gui_Alta_relaciones_v2.java.*
  - *Gui_optimizar_v1.java*
  - *ModeloTablaAtributos.java*
  - *ModeloTablaRelaciones.java*

3.5.3 Programación de las interfaces gráficas.

De manera general todas las interfaces son clases que extienden de la clase JDialog.

Hemos proporcionado accedentes y mutadores a todos los elementos de las interfaces que tuvieran que ser modificado o accedidos.

- ✓ **presentacion**
  - **Gui_Principal.java**

En la **parte superior** hemos situado un menú en el que aparecen las distintas opciones de la aplicación. Hemos programado cada uno de los menús con las distintas opciones, estableciendo que opciones tienen que estar activas al inicio, y cuando se tienen que activar las restantes.

Este menú esta formado por *JMenuItem*, hemos definido ActionListeners para estos *JMenuItem*, y es en estos ActionListeners donde programamos lo que tiene que suceder cuando el usuario pulsa cada una de las opciones del menú.

```
JMenuBar jmb = new JMenuBar();
…
JMenuItem  jm_nuevo_Diagrama = new JMenuItem("Nuevo
Diagrama",'N');


…

jm_nuevo_Diagrama.addActionListener(new ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
     Gui_Alta_Diagrama  gui00 = new Gui_Alta_Diagrama();
     gui00.setVisible(true);
   }
});//----------- fin addActionListener
```

En el ejemplo vemos como cuando el usuario pulse el botón "Nuevo Diagrama", el evento será recogido por su ActionListener que será el encargado de lanzar la interfaz de Alta de diagramas y de ocultar la interfaz principal.

En la **parte central** tenemos una zona de consultas, que permanecerá oculta hasta que no se cree o se abra un diagrama.

En la parte superior tenemos un panel que nos muestra el nombre del diagrama, un combobox con todas las entidades del diagrama, y otro combobox con todas las relaciones del diagrama.

Para rellenar los combobox hemos creado funciones que los rellenan a partir de un ArrayList de TEntidades ó TRelaciones respectivamente.

Debajo de este panel hemos situado dos tablas que muestran información de las Entidades y de las Relaciones. Para crear la tabla de Entidades hemos creado un modelo de la tabla de entidades:

```
private ModeloTablaEntidades modelo_entidad = new
ModeloTablaEntidades();
```

En este modelo hemos programado las características de la tabla, que filas tiene la tabla, que celdas son editarles, y métodos como *anadirEntidad, eliminarEntidad, vaciarTabla(), hayEntidad().*

```
public class ModeloTablaEntidades extends DefaultTableModel
{
 …

public ModeloTablaEntidades()
// Accion: Define la tabla de Entidades

public void AnadirEntidad(Object[] entidad)…
// Accion: añade una entidad a la tabla, es decir una fila entera

public void eliminarEntidad(int posición)
// Accion: eliminar una entidad de la tabla

public void VaciarTabla()
// Accion: vaciar la tabla

public boolean hayEntidad()
// Accion: true → hay alguna entidad, false → e.o.c

public boolean isCellEditable(int row, int col)
// Accion: Determinar que celdas de la tabla con editables

}
```

De manera similar para crear la tabla de Relaciones hemos creado previamente un modelo de la tabla de relaciones.

Hemos creado también funciones que se encargan de poblar los modelos de las tablas con datos, a partir de un ArrayList de TEntidades o TRelaciones respectivamente.

Hemos programado dos funciones *actualiza_entidades(),* y *actualiza_relaciones()* que se encargan de actualizar los combobox y tablas de entidades y relaciones respectivamente.

```
public void setComboBox_entidades(JComboBox combo,
ArrayList<TEntidad> lista)
…

public void setComboBox_relaciones(JComboBox combo,
ArrayList<TRelacion> lista)
…

public void setModeloTable_entidades(ModeloTablaEntidades modelo,
ArrayList<TEntidad> lista)
…

public void setModeloTable_relaciones(ModeloTablaRelaciones
modelo, ArrayList<TRelacion> lista)
…

public void actualiza_entidades()
…

public void actualiza_relaciones()
```

En la **parte inferior** hemos programado un editor de mensajes, que se utiliza para mostrar mensajes de información al usuario. Además hemos definido diferentes estilos para este editor, por ejemplo *estilo Azul* para mostrar los mensajes con color azul cuando todo ha ido bien y *estilo Rojo* para mostrar los mensajes con color rojo cuando algo ha ido mal.

```
private JTextPane creaEditor()
{
 …
}

private StyleContext creaEstilos()
{
//--- Define los estilos del panel de mensaje de operaciones
…
}
```

- ✓ **presentacion.archivo:**
  - **GuiAltaDiagrama.java**

En este formulario hemos situado los campos de edición necesarios para que usuario pueda introducir el nombre, el usuario y el password.

Hemos programado accedentes y mutadores para los campos de edición.

Al pulsar el botón aceptar, en el ActionListener de este botón se valida que el campo nombre este cumplimentado, (para ello hemos creado la función **nombre_cumplimentado()** que devuelve verdadero si el campo Nombre contiene un valor y falso en caso contrario).

Si hay nombre, extrae los valores de los campos utilizando sus accedentes correspondientes, crea un transfer de *TEsquema* con esos valores, y lanza el evento **ALTA_DIAGRAMA** al **controlador** pasándole el transfer de TEsquema creado. Si no hay nombre, se lanza un mensaje al usuario explicándole que el campo nombre es obligatorio.

```
//--- Funciones
public boolean nombre_cumplimentado() …

//--- Accedentes
public String getCampoNombre() …
public String getCampoUsuario() …
public String getCampoPassword() …

//--- Mutadores
public void setCampoNombre(String s) …
public void setCampoUsuario(String s) …
public void setCampoPassword(String s) …
```

- **GuiAbrirDiagrama.java**

Esta interfaz es prácticamente igual que la GuiAltaDiagrama.java. Hemos sustituido el campo de edición Nombre, por un combobox donde aparecen los nombres de todos los diagramas existentes en la Base de Datos.

```
private JComboBox comboBox_diagramas = new
                                       JComboBox();
```

Para mostrar el listado de diagramas en el combobox, hemos creado la función **setComboBox**.

```
public void setComboBox(JComboBox combo,
                        ArrayList<TEsquema> lista)
```

Esta función accede a cada uno de los elementos que del ArrayList que le pasamos como parámetro, y para cada uno de estos elementos accede a su nombre (en este caso, el nombre del Esquema), y lo añade en el combobox que también pasamos como parámetro.

Y para acceder al valor del combobox seleccionado hemos creado su accedente correspondiente: p*ublic String*

dame_nombre_diagrama() que nos devuelve el ítem seleccionado en dicho combobox.

Entonces para cargar y actualizar los valores del combobox con el nombre de los diagramas, en el código llamamos a esta función pasándole como parámetros el combobox que queremos que actualice, y un ArrayList de TEsquema (Un lista con todos los esquemas).

```
setComboBox(comboBox_diagramas,Gui_Principal.ListaEsquemas);
```

El resto de la programación es similar al de la guiAltaDiagrama, al pulsar el botón aceptar, se acceden a los valores de los campos Nombre, Usuario, password a través de sus accedentes. Creamos el transfer de Esquema, y lanzamos al controlador el evento **DIAGRAMA_ABIERTO** junto con el transfer del esquema creado.

- ✓ **presentacion.entidades**
  - **Gui_Alta_Entidad_v3.java**.

En la **parte superior** hemos situado campos de edición para introducir el nombre de la entidad y su descripción.

En la **parte central** una tabla que muestra los atributos que tiene la entidad. Esta tabla se basa en un Modelo de tabla de atributos, en el que hemos definido las características de las columnas, las celdas de la tabla, funciones para añadir un atributo, para eliminarlo, … etc.

```
public class ModeloTablaAtributos extends DefaultTableModel
{
 public ModeloTablaAtributos()
 //--- Define la Tabla de Atributos
 // TABLA: 0-Nombre|1-Tipo|2-Longitud|3-Clave|4-Unsigned|
           5-Not  Null|6-Default

public void AnadirAtributo(Object[] atributo)
// Accion: añade un atributo a la tabla, es decir una fila entera

public void eliminarAtributo(int posicion)
       // Accion: eliminar un atributo de la tabla
public boolean hayAtributo()

public boolean isCellEditable(int row, int col)
// Accion: true → hay algún atributo, false → e.o.c

public boolean isCellEditable(int row, int col)
// Accion: Determinar que celdas de la tabla con editables

…
}
```

En el panel donde está la tabla de atributo también hemos situado dos botones:

- "Eliminar Atributo" → al pulsarlo su *ActionListener* elimina el atributo seleccionado de la tabla

- "Añadir Atributo" → al pulsarlo su *ActionListener* muestra el editor de atributos.

En el **parte inferior** hemos situado el editor de atributos, tiene un campo para introducir el nombre del atributo, y un combobox con un listado de tipos para que el usuario seleccione el tipo del atributo, este combobox tiene un *ActionListener* que en función del valor del tipo seleccionado en el combobox muestra o no ciertos campos del editor. Por ejemplo si el tipo es *"Varchar"* muestra el campo para introducir la longitud, pero si el tipo es "Date" oculta dicho campo.

En dicho editor aparecen también casillas de verificación, para establecer propiedades del atributo, si es *clave*, si es *Not Null*, o si es *unsigned*, y también campos para introducir *valores por defecto* del atributo ó *longitudes*.

En dicho panel aparecen dos botones:

- "Limpiar Campos"→ su *ActionListener* vacía los valores de los campos del editor de atributos.
- "Añadir Atributos" → Su *ActionListener* revisa que el campo nombre atributo tenga algún valor, y en ese caso recoge la información de los campos del editor, y los guarda en el modelo de la tabla de atributos, y oculta el panel de edición de atributos.

Finalmente al pulsar el botón "Aceptar" del formulario, se revisa que se halla introducido el nombre de la Entidad, y al menos un atributo con clave. En ese caso se recogen los datos del nombre, descripción y los atributos introducidos haciendo un recorrido del modelo de la tabla Atributos. Y se le pasa al controlador el evento **ALTA_ENTIDAD** junto con un transfer de la Entidad, y una lista con los atributos introducidos.

```
Controlador.getInstance().accion(EventoNegocio.ALTA_ENTIDAD,
                                 tentidad,
                                 listaAtributos);
```

- ✓ **presentacion.opciones**
  - **Gui_conexiones_BD.java**

En esta interfaz hemos incluido campos de edición para que el usuario pueda introducir el Nombre, Usuario y Password de la Base de datos a la que se quiere conectar.

Hemos introducido un botón "Limpiar" al pulsarlo su ActionListener, "limpia" el valor de los campos de datos, para ello accede a los mutadores que hemos creado de los campos de edición y establece que su valor es el de una cadena vacía.

Tenemos también un botón "Test", al pulsarlo su ActionListener lanza el evento **TEST_CONEXION_BD** al controlador (con este evento comprobara se comprobara el estado de la conexión a la Base de Datos).

En el ActionListener del botón "Aceptar", primero hemos validado que se han introducido los campos obligatorios (para ello hemos creado las funciones auxiliares *nombre_DB_Cumplimentado()*, *nombre_User_Cumplimentado(),nombre_Password_Cumplimentado*(), que nos devuelven verdadero si el valor de estos campos es distinto al de una cadena vacía). En el caso que de que los campos de edición hayan sido cumplimentados, extraemos sus valores usando sus accedentes, y con estos valores creamos un transfer de *TConexion*. Y finalmente le lanzamos al controlador el evento *ALTA_CONEXION_BD*, junto con el transfer de conexión creado.

```
//--- funciones
public boolean nombre_DB_Cumplimentado() …
public boolean nombre_User_Cumplimentado()
public boolean nombre_Password_Cumplimentado()
```

- ✓ **presentacion.relaciones**

  - **Gui_Alta_relaciones_v2.java**.

    Esta interfaz es similar a la interfaz de Alta de entidades, tiene campos de edición para introducir el nombre de la relación y su descripción, y un editor de atributos que es el mismo que utilizamos en el formulario de Alta de entidades.

    En esta interfaz aparece un grupo de botones que permiten seleccionar el tipo de relación (Binaria, Ternaria, Es un).

    En la parte central hemos situado tres paneles en tres columnas que permiten definir el nombre de la entidad, su participación y cardinalidad.

    Para definir el **nombre de la entidad** hemos colocado tres combobox en los que cargamos los nombres de las entidades existentes en el diagrama.

    Para definir la **participación** hemos colocado tres combobox en los que se puede seleccionar "total" o "parcial".

Para definir la **cardinalidad** tenemos tres combobox que en función del tipo de relación seleccionado, permitirá seleccionar "1,N" ó "1..N, N,M".

El ActionListener de los botones del tipo de relación se encarga de mostrar u ocultar los campos apropiados en cada caso.

- El botón "Es un" muestra solo dos combobox para seleccionar cual es la entidad madre y cual la entidad hija.

- El botón "Binaria" muestra una filas de combobox para definir el nombre, participación y cardinalidad de la Entidad 1, y una segunda fila de combobox para definir el nombre, participación y cardinalidad de la Entidad 2, muestra la tabla de Atributos y activa el editor de atributos.

- El botón "Ternaria" muestra las tres filas de combobox para introducir el nombre, participación y cardinalidad de tres entidades, muestra la tabla de Atributos y activa el editor de atributos.

Al pulsar el botón "Aceptar" si hay nombre de la relación definido, en función del tipo de relación definido se guardan los datos introducidos por el usuario y se lanza el evento correspondiente junto con sus datos.

Por ejemplo si es una relación "Es un":

Se crea un transfer de relación "Es Un", se guardan en el los identificadores de la entidad madre e hija, el identificador del esquema actual y se lanza al controlador el evento ALTA_RELACION_ES_UN junto con el transfer de relación "es un".

```
Controlador.getInstance().accion(EventoNegocio.ALTA_RELACION_ES_UN
                                 ,trelacion_es_un);
```

Si es una relación "binaria" o "ternaria":

Se crea un transfer de Relación en el que guardamos el nombre de la relación, la descripción y el identificador del esquema actual.
Creamos una lista de entidades con los valores de nombre, participación y cardinalidad que tenemos para cada entidad.

Creamos una lista de atributos con los valores de los atributos que existan en el modelo de la tabla de atributos.
Se lanza al controlador el evento ALTARELACION junto con el transfer de relación, la lista de Entidades y la lista de Atributos.

```
Controlador.getInstance().accion(EventoNegocio.ALTA_RELACION,
                                 trelacion,
                                 listaEntidades,
                                 listaAtributos);
```

- **Gui_optimizar_v1.java**

Este formulario recibe como parámetros:

```
codigo_tipo_relacion → Tipo de relación
                       (1-Binaria, 2-Ternaria, 3- Es_un)
id_relacion → identificador de la relación
id_entidad1, id_entidad2 → identificador de la Entidad 1 y
                           Entidad 2
 participacion1, participacion2 → participación de la Entidad 1 y
                                  Entidad 2
                                  (1 – Total , 2 - Parcial)
 cardinalidad1,cardinalidad2 → cardinalidad de la Entidad 1 y
                               Entidad 2
                               (1- Uno, 2- Muchos)
```

En la parte superior de la interfaz hemos situado una etiqueta para que muestre el tipo de relación que estamos optimizando: *JLabel label_tipo_relacion*, en el código hacemos un *switch* del valor del codigo_tipo_relacion si es Binaria ponemos en la etiqueta: "Optimizar Relacion Binaria", si es Ternaria: "Optimizar Relación Ternaria".

Debajo de esta etiqueta hemos situado un panel que va a contener un *ButtonGroup* con las preguntas que vamos a ofrecer al usuario:

```
   private ButtonGroup grupoBotonesTipoOptimizacion = new
                       ButtonGroup();
```

Y hemos creado RadioButtons para cada una de las preguntas de optimización que se le pueden ofrecer al usuario.

```
private JRadioButton boton_optimizacion0 = new JRadioButton("Cada
vez que realice consultas respecto a esta relación le interesará
sólo la información referenciada por los atributos claves de las
entidades que relaciona.",true);
```

…boton_optimizacion1 …boton_optimizacion7

Ahora en el código miramos la cardinalidad de la entidad 1 y de la entidad 2, y distinguimos 3 casos: a) Uno a Uno, b) Uno a Muchos, c) Muchos a Muchos.

Después para cada uno de estos casos miramos la participación de las entidades.

En este punto, sabiendo el tipo de relación, el caso, y la participación de las entidades, agregamos al panel los botones con las preguntas de optimización que corresponden al caso en cuestión.

Por ejemplo:

Si la relación es Binaria, caso: Uno a Uno, y la participación de ambas es Total, se añadirían los botones: *boton_optimizacion0, boton_optimizacion1* y *boton__optimizacion2.*
Es decir al usuario solo se le presentan las preguntas de optimización que corresponden al caso.

Otro ejemplo, si en este mismo caso, la participación fuese Parcial de alguna entidad los botones añadidos serían los botones de optimización 0, 4, 5, 6 y 7.

Hemos creado ActionListeners para estos botones, y hemos programado que cuando uno de estos botones sea pulsado guarde el valor del tipo de optimización en la variable **codigo_optimizacion_final.**

Ejemplo: Relación Binaria Uno a Uno con Participación Parcial de alguna entidad.

Pregunta:" Cada vez que realice consultas …"
(Boton0) → si se pulsa → codigo_optimizacion = -1

Pregunta:" Realizará consultas y querrá obtener información adicional de E1"
(Boton4) → si se pulsa → codigo_optimizacion = 4

Pregunta:"Realizará consultas y querrá obtener información adicional de E2"
(Boton5) → si se pulsa → codigo_optimizacion = 5

Pregunta:"Estima que la entidad E1 tendrá más información que la E2."
(Boton6) → si se pulsa → codigo_optimizacion = 6

Pregunta:" Estima que la entidad E2 tendrá más información que la E1"
(Boton7) → si se pulsa → codigo_optimizacion = 7

En la variable **codigo_optimizacion_final** tenemos guarda la selección de optimización del usuario.

Ahora cuando el usuario pulse el botón "Aceptar" su *ActionListener*, en función del **codigo_optimizacion_final** (la selección del usuario), lanzará al controlador el evento correspondiente al tipo de optimización que hay que aplicar, junto con los identificadores de la relación y las entidades.

Por ejemplo si el usuario selecciona la pregunta del botón 4:

Se lanza el evento:

```
Controlador.getInstance().accion(EventoNegocio.NO_CREA_REL_1_1_PARCIAL,
identificador_entidad2, identificador_relacion,
identificador_entidad1);
```

De esta forma hemos conseguido una única interfaz gráfica de optimizaciones para todos los tipos de relación, casos de participación y cardinalidad.

3.6 Capa de Acceso a Datos.

- Es la capa destinada a acceder a los datos, mediante el uso de *Data Access Objects (DAOs).*

- Los datos están guardados en una sistema de representación relacional, por lo tanto los *DAOs* se encargar de obtener representaciones orientadas a objetos (*Transfers)* que serán usados por los servicios de la lógica.

- Con el uso de esta capa nos aseguramos la independencia entre la lógica del negocio y el sistema de representación de los datos, pudiendo cambiarse la representación de los datos sin afectar a la lógica de nuestra aplicación.

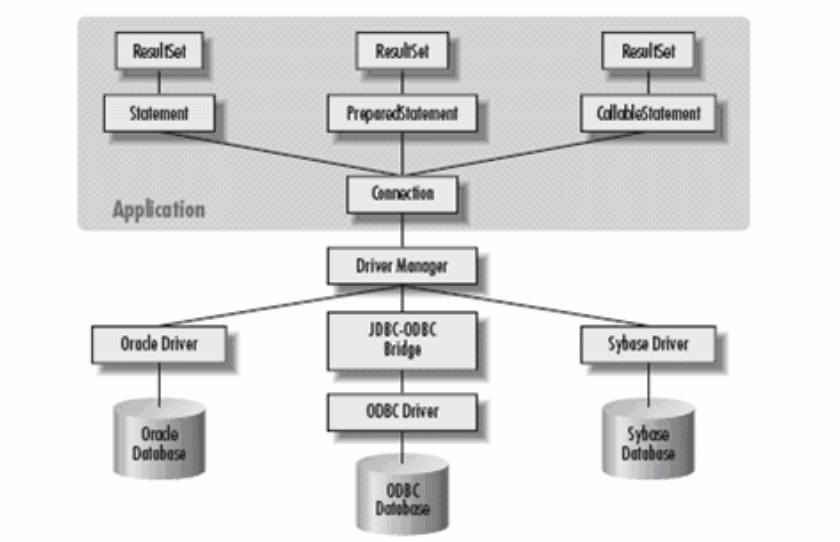
- Usaremos tantos DAOs como módulos existen en nuestro sistema: Diccionario de Datos y Generador de Modelo Relacional. La creación de ellos la hacemos a través de una *Factoría Abstracta* implementada como un *Singleton.*

- El acceso genérico a los datos puede verse en el siguiente diagrama de secuencia:

- Para ejecutar consultas *SQL* en bases de datos relacionales, usamos *Java DataBase Connectivity (JDBC)* y como Sistema Gestor de Bases de Datos *MySQL.*

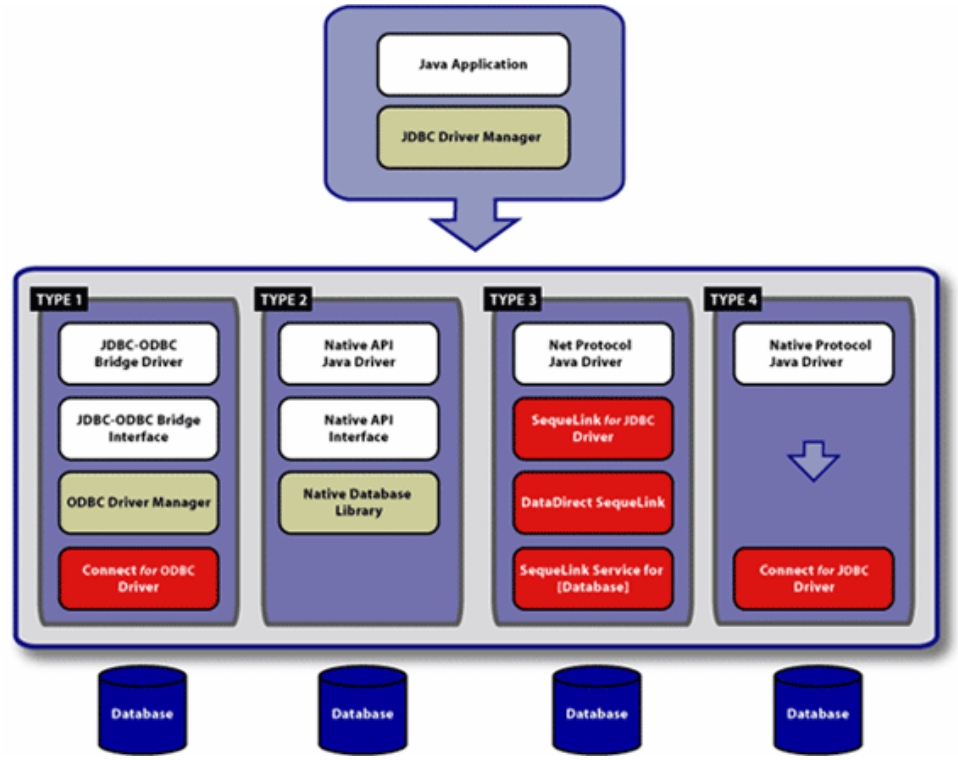
- La API *JDBC* es independiente del SGBD, esta distribuida dentro de los paquetes java.*sql* y javax.sql.

-La API ofrece las clases e interfaces para establecer una conexión a una base de datos, ejecutar una consulta y procesar los resultados:

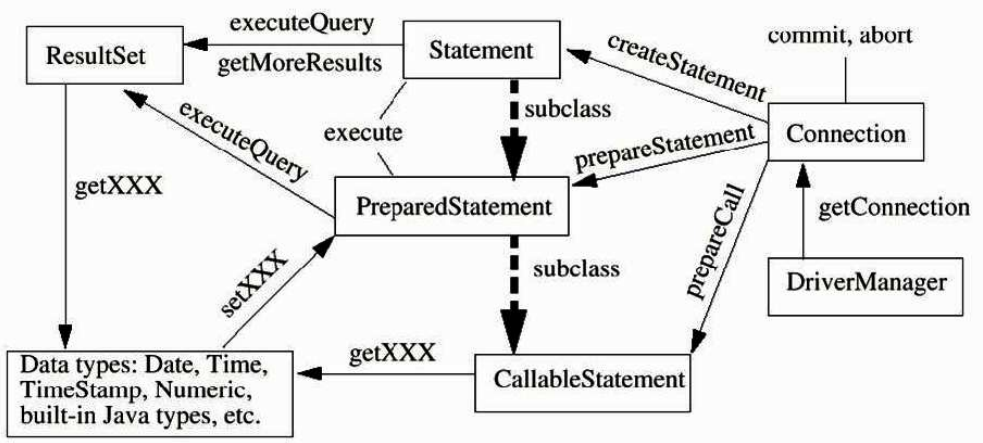


- Para acceder a la base de datos es necesario un *driver*, conjunto de clases e destinadas a implementar las interfaces del API y acceder a la base de datos. Existen cuatro tipos de *drivers* de conexión:
1. Tipo I: puente JDBC-ODBC
2. Tipo II: driver Java parcial
3. Tipo III: driver Java puro
4. Tipo IV: driver Java puro para conexión directa a SGBD

- Emplearemos el driver de tipo IV *MySQL Connector/J*, debido a la compatibilidad en diversos entornos y a la facilidad de uso, ya que no es necesaria ninguna configuración en el sistema operativo.



- El funcionamiento de JDBC puede verse a través del siguiente diagrama:



3.6.1 Estructura de paquetes de la capa de datos

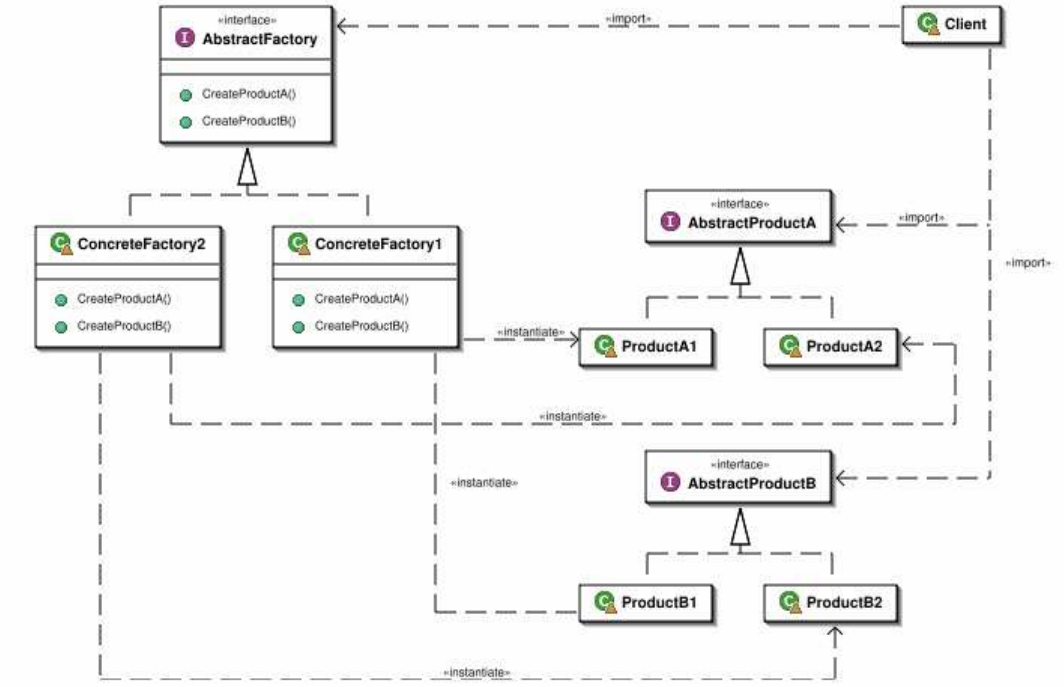
Integración.

- *DAOFactoria*
  Es la factoría abstracta de DAOs.

Dado un conjunto de clases abstractas relacionadas, el patrón *Abstract Factory* permite el modo de crear instancias de estas clases abstractas desde el correspondiente conjunto de subclases concretas. Proporciona una interfaz para crear familias de objetos relacionados o
dependientes sin especificar su clase concreta.

El patrón *Abstract Factory* puede ser muy útil para permitir a un programa trabajar con una variedad compleja de entidades externas, tales como diferentes sistemas de ventanas con una funcionalidad similar.

Funcionamiento de la factoría abstracta:



La clase DAOFactoria contiene como único atributo, *unicaInstancia* de tipo DAOFactoria que sólo se creará una vez y cada vez que se le solicite la devolverá.

A través de ese atributo devuelto se podrá acceder a los DAOs, que en consecuencia también sólo se crearán una vez y no andaremos creando un mismo DAOs cada vez que lo necesitamos. Así que, además consta de métodos abstractos que crean cada uno de los DAOs que usamos en nuestra aplicación (*creaDAODiccionario* y *creaDAOGenerador*).

- *DAOFactoriaSQL*

Hereda de la clase DAOFactoria e implementa los métodos abstractos que hay en ésta llamando a las constructoras de los DAO específicos de cada clase.

Para el resto de paquetes que forman integración tendremos un interfaz (*DAODiccionario* y *DAOGenerador*) que contiene los prototipos de los servicios que proporciona ese DAO al que hace referencia el nombre y la clase que los implementa (*DAODiccionarioImp* y *DAOGeneradorImp*):

En todos los paquetes se encuentran declaradas las tablas que se necesitan para cada módulo y los métodos necesarios para operar sobre ellas.

Integración.Diccionario.
DAODiccionario y DAODiccionarioImp

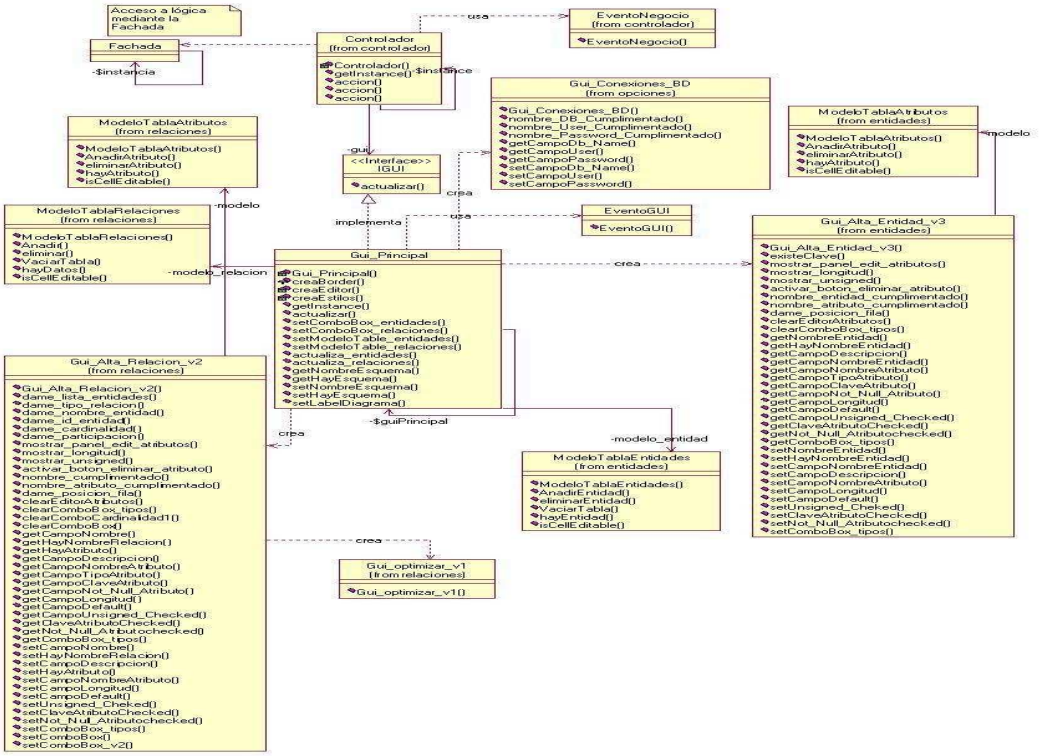
Almacena toda la información relativa al diccionario de datos tal y como se vio en su apartado.

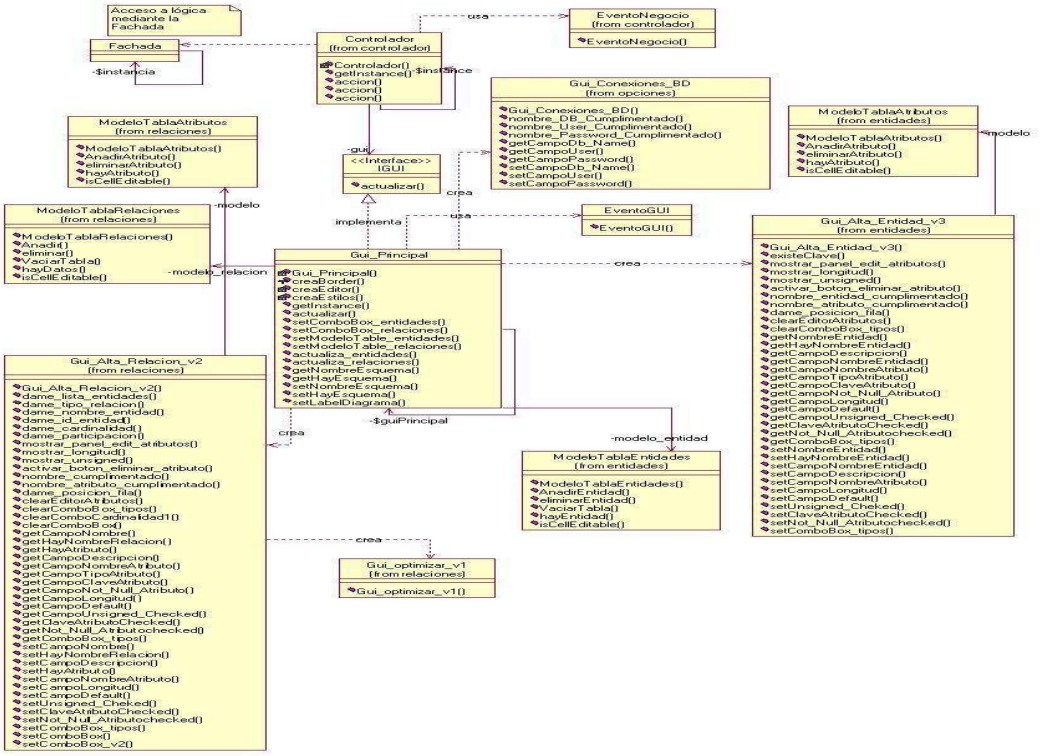
Integración.Generador
DAOGenerador y DAOGenerador

Contiene la información referida al Generador del Modelo Relacional, contiene los métodos necesarios para crear un nuevo esquema y almacenarlo en el SGBD MySQL

3.7 Diagramas de Clases

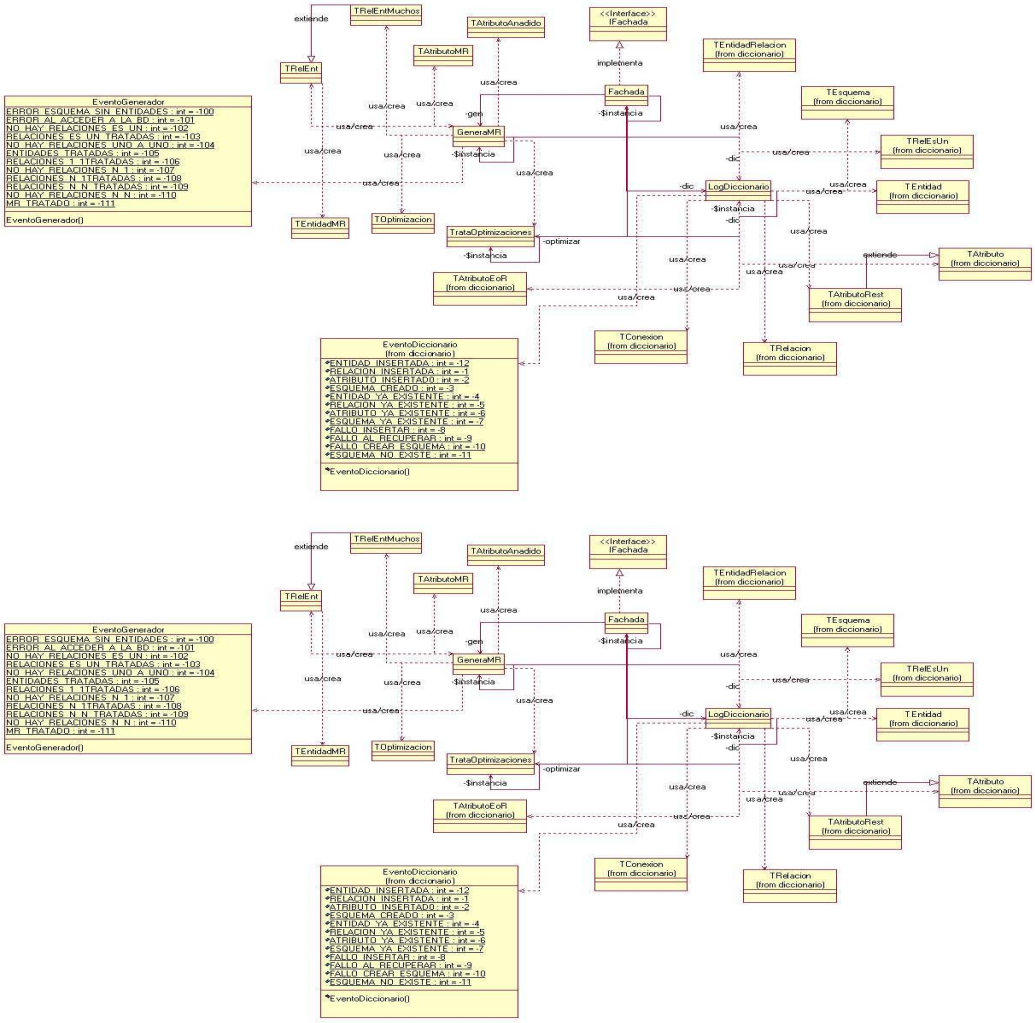
3.7.1 Capa de Presentación



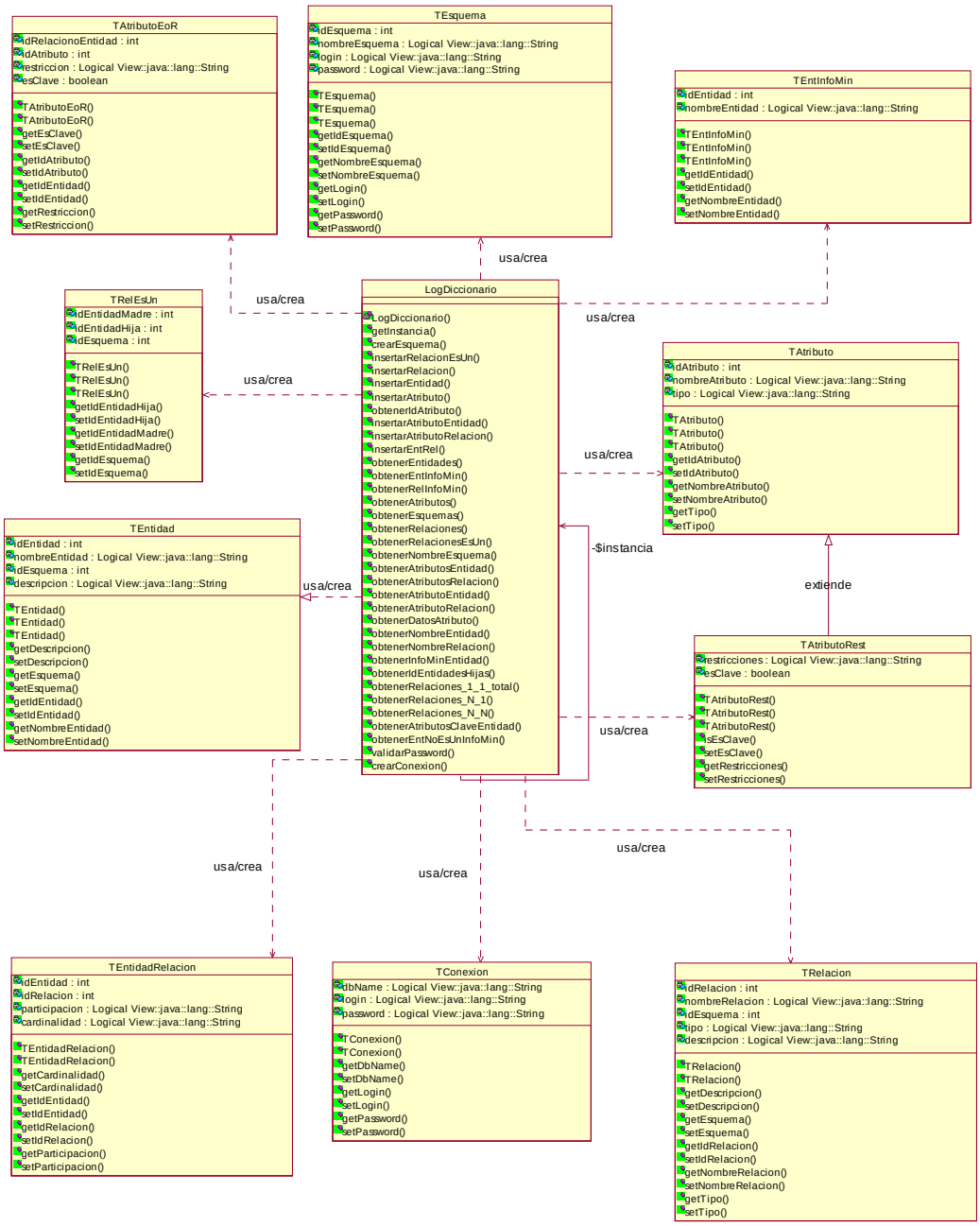


3.7.2 Capa Lógica

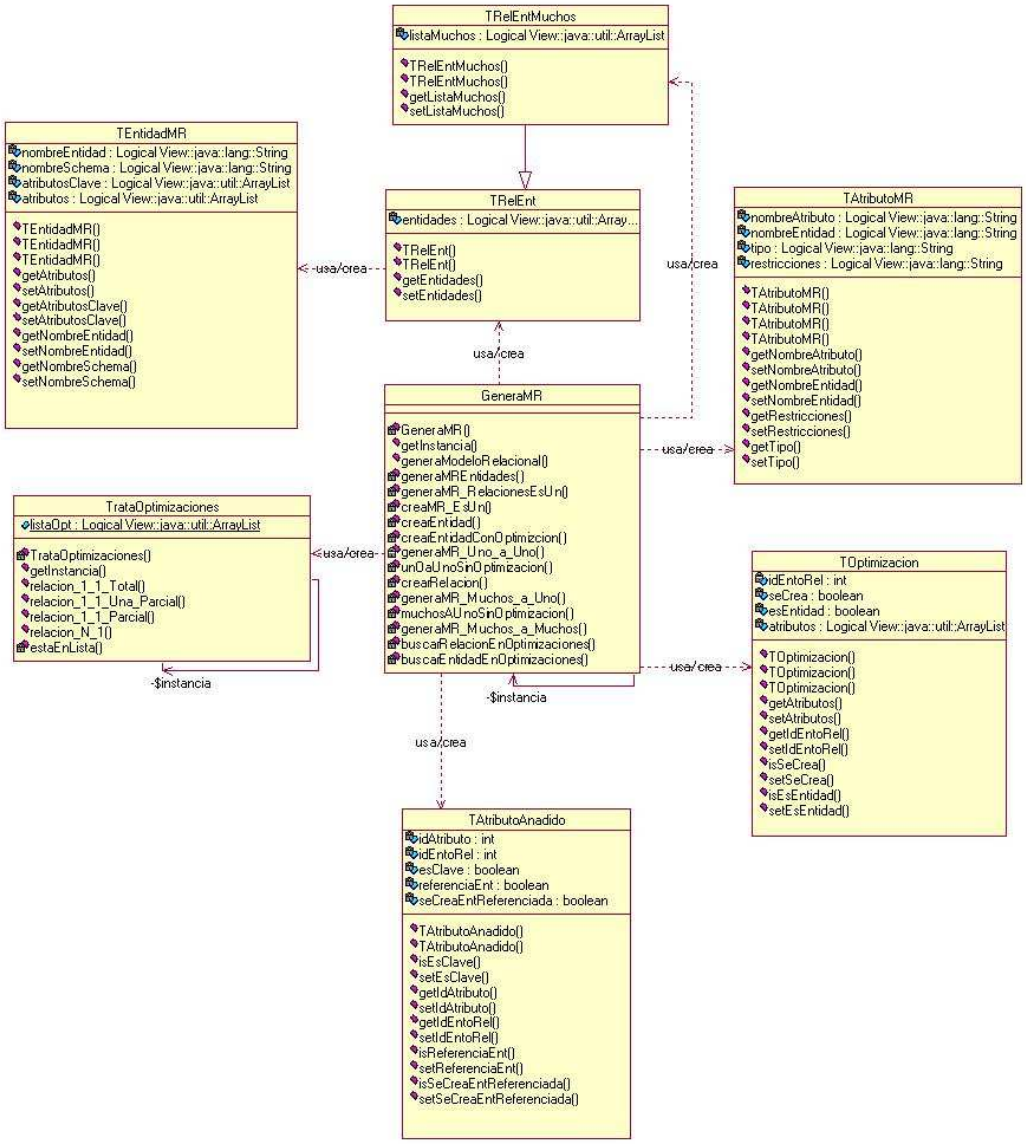
3.7.2.1 Visión general



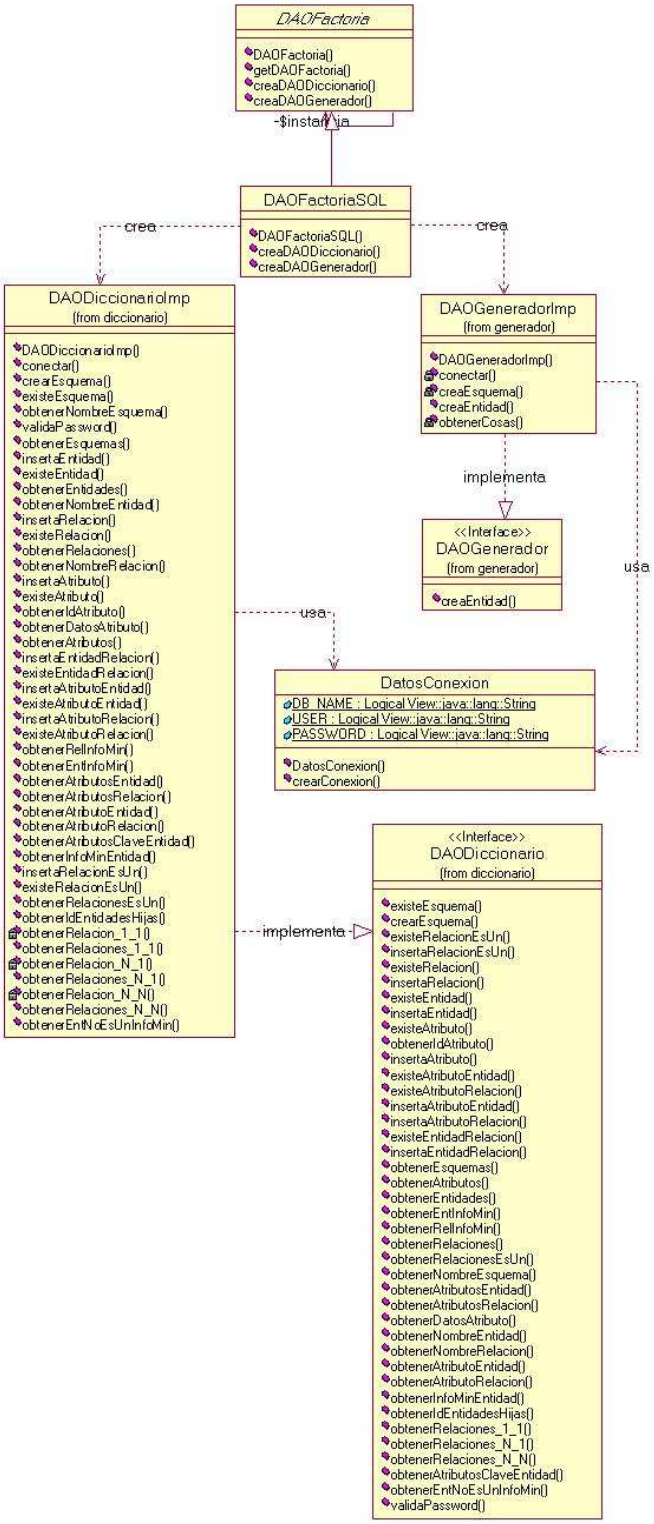
3.7.2.2 Diccionario de datos



3.7.2.3 Generador del modelo relacional



3.7.3 Capa de Acceso a datos



3.8 Optimizaciones en el Modelo Relacional

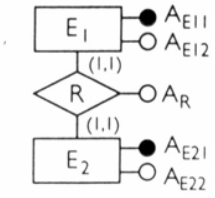
Al introducir una relación, el sistema, atendiendo al tipo de la misma y a la participación de las entidades que une, le ofrecerá al usuario una serie de mejoras a aplicar sobre el modelo relacional que represente a dicha relación. A continuación se expone cuando serán más convenientes unas opciones u otras y por qué:

- ✓ **Relaciones binarias uno a uno…**

  Desde un punto de vista semántico una relación de este tipo se traduce como que un ejemplar de la entidad E1 sólo se podrá relacionar con uno de la entidad E2 y viceversa.

  El modelo relacional que se suele asumir en estos casos por lo general es: E1 y E2 se crean normalmente, y R(A_R, A_{E11},A_{E21}), donde la clave de R es o bien la de E1 o bien la de E2 (es indiferente cuál se seleccione de las dos).

  - *...con participación total para ambas entidades:* Todas las instancias de E1 tendrán que estar relacionadas con una de E2 y viceversa.

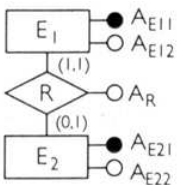


  Como alternativa se propone E1(A_{E11}, A_{E12},A_{E21}, A_R), es decir, meter en E1 la relación R. Esto conviene cuando se realicen un número considerable de consultas sobre la relación y se quiera obtener no sólo la información básica sobre E1 sino otra más concreta (o sea, no sólo la clave sino también otros atributos de interés).

  Si no lo hacemos así cada vez que realizásemos una consulta sobre R, habría que llevar a cabo otro acceso a la base de datos para recuperar los datos de E1.

  Esta alternativa se puede ofrecer análogamente pero cambiando E1 por E2.

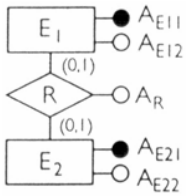
  - *...con participación parcial en una de las entidades:* En este caso todas los ejemplares de E1 han de estar relacionados con uno de E2, sin embargo, en E2 pueden existir instancias que no se relacionen con ninguna de E1.



La única posibilidad es introducir la relación en la entidad afectada por la participación total, en el ejemplo sería: E1(A_{E11}, A_{E12},A_{E21}, A_R). Esto no se aplica para E2 ya que como no todas sus tuplas están relacionadas con E1 habría muchas que tendrían el campos de la clave de E1 a null con lo cual se desperdiciaría memoria.

Se ha de tomar este camino cuando la mayoría de las veces que realicemos una consulta sobre R queramos obtener información adicional sobre ella y no meramente su clave.

▪ *...con participación parcial para ambas entidades:* En este caso ni a todas las instancias de E1 le corresponde una de E2, ni a todas las de E2 una de E1.



Podemos o bien introducir la relación en E2 o bien en E1, mas en ambos casos podrán existir tuplas cuyos campos introducidos a causa de la relación estén vacíos. Con lo cual, a igualdad de datos adicionales a obtener tanto de una entidad como de otra, será mejor elegir aquella cuyo número de instancias sea menor para así desaprovechar la menor cantidad de memoria posible.

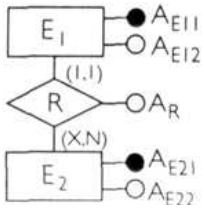
Esto se debe elegir si se consulta con mucha frecuencia la relación existente entre las entidades involucradas y además en esas consultas se quiere obtener información sobre las entidades que no sólo atañen a su clave.

✓ **Relaciones binarias de muchos a uno...**

Desde un punto de vista semántico una relación de este tipo se traduce como que un ejemplar de la entidad E1 (la afectada por el "muchos") sólo se podrá relacionar con una de la entidad E2 pero podrán existir otros ejemplares de E1 que se relacionen con ese mismo individuo de E2.

El modelo relacional que se suele asumir en estos casos por lo general es: E1 y E2 se crean normalmente, y R(A_R, A_{E11},A_{E21}), donde la clave de R es la de E1.

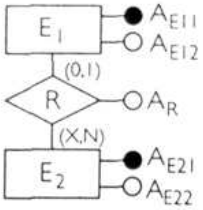
▪ *...con participación total:* Todas las instancias de E1 están relacionadas con una de E2.



Entonces puede ser conveniente introducir la relación existente entre ambas entidades en el modelo relacional de la entidad E1 siempre que el número de consultas sobre R sea elevado y se

quiera obtener información adicional sobre las características de los individuos de E1.

▪ *...con participación parcial:* En E1 puede haber ejemplares que no estén relacionados con ninguno de E2.

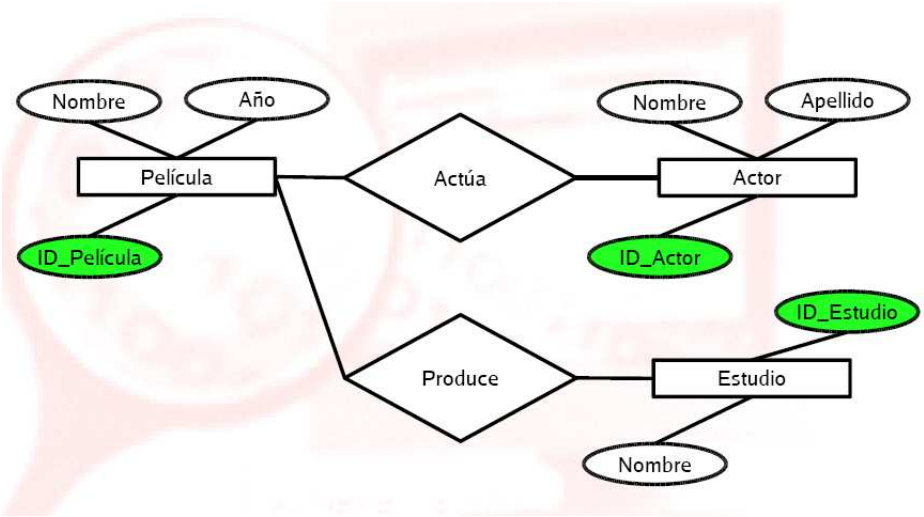


La alternativa al modelo relacional general sería igual que en el caso anterior pero en este caso habría que tener en cuenta que existirán tuplas cuyos campos sean vacíos. Con lo cual esta decisión conviene tomarla cuando, a pesar de que no todas las instancias de E1 estarán relacionadas con las de E2 un gran número si lo estará.

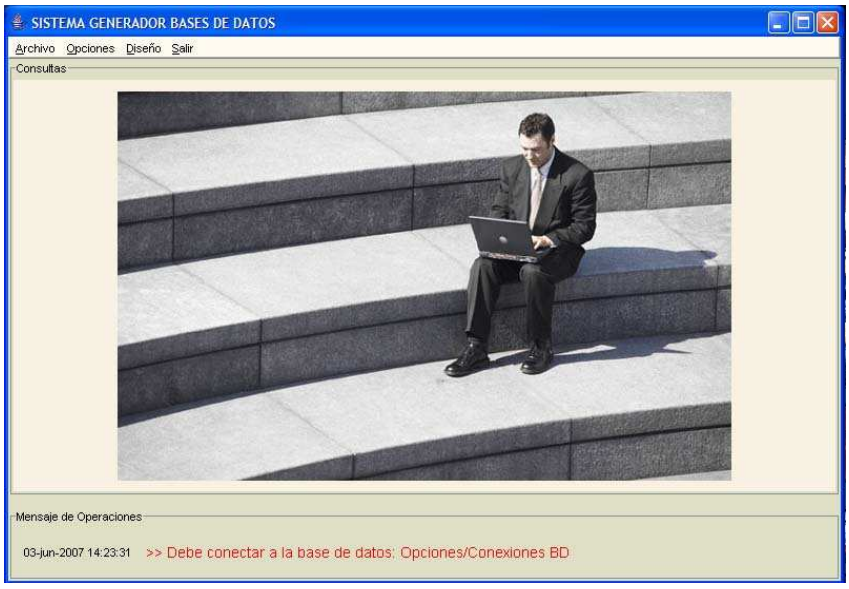
4. RESULTADOS OBTENIDOS

En este apartado queremos valorar si nuestro proyecto ha cumplido los objetivos propuestos en su realización.

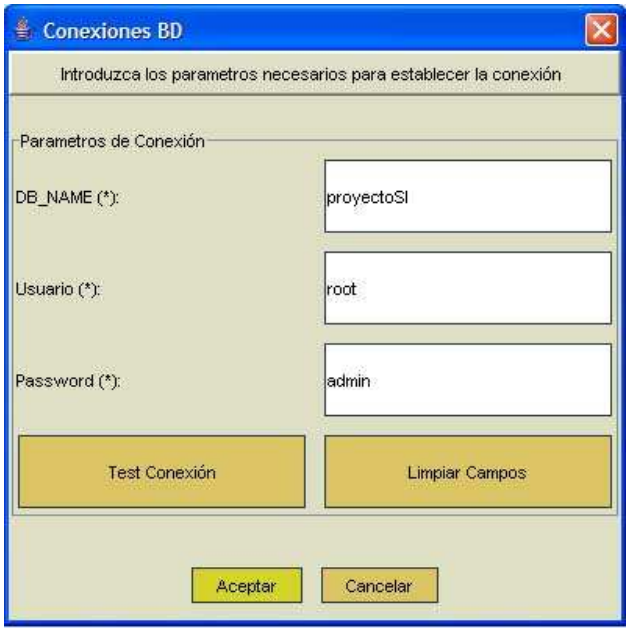
Para ello hemos utilizado la aplicación "Sistema Generador de Bases de Datos" para generar la base de datos del ejemplo que propusimos en el Documento de explicación informal del proyecto.



Accedemos a la aplicación.

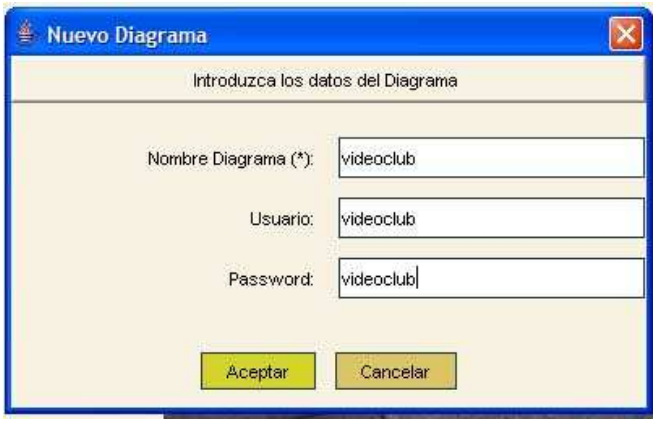


Configuramos la conexión a la base de datos. (Opciones/Conexión BD)



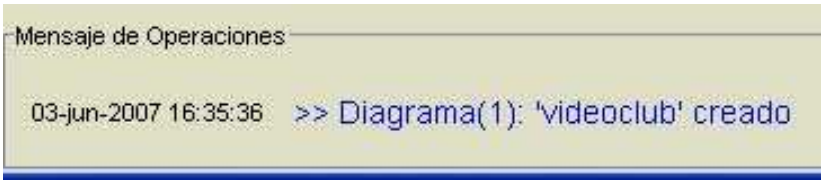
La conexión se establece correctamente.

Procedemos a crear el nuevo diagrama Videoclub: (Archivo/Nuevo Diagrama)

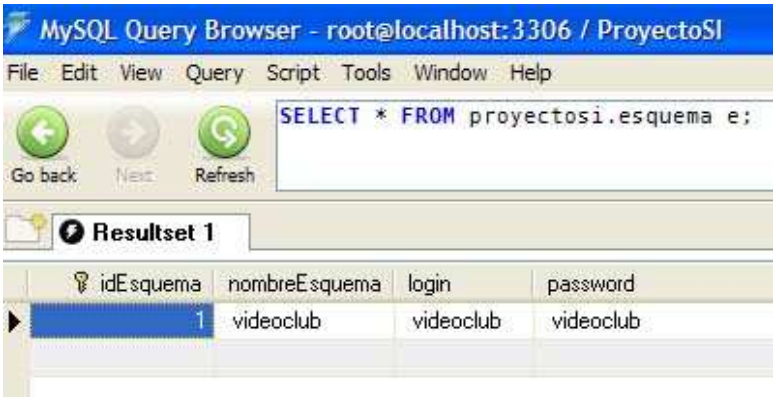


Queremos crear el diagrama videoclub, y asignándole un usuario y un password.

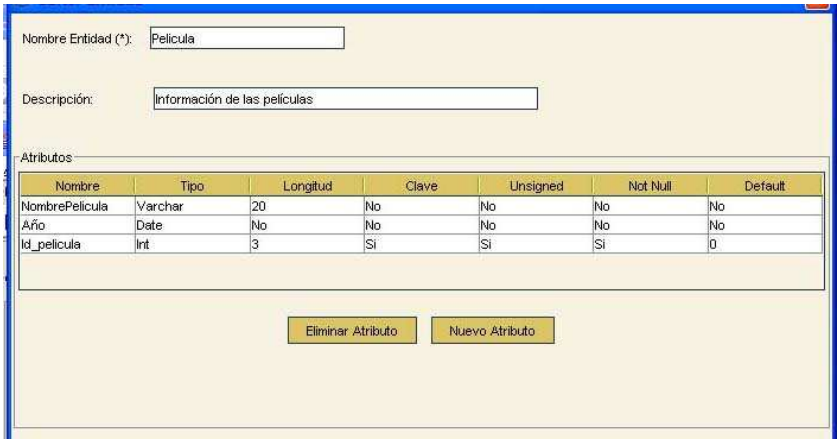
El diagrama se crea correctamente.



Comprobamos que los datos de dicho esquema se han guardado correctamente en el diccionario de datos:



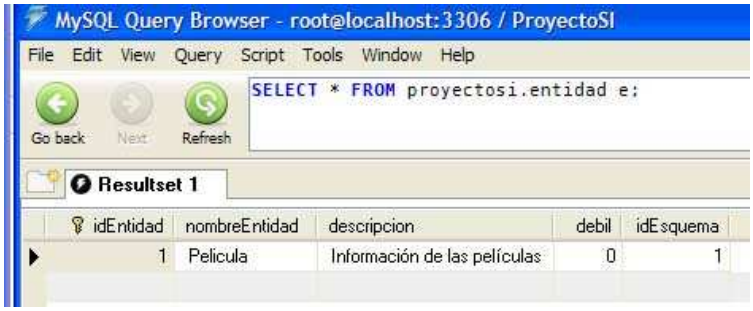
Ahora creamos la entidad Película. (Diseño/Entidades)



Introducimos cada uno de sus atributos, mediante el panel de edición de atributos.
Pulsamos el botón Aceptar, y vemos como la entidad se crea correctamente.

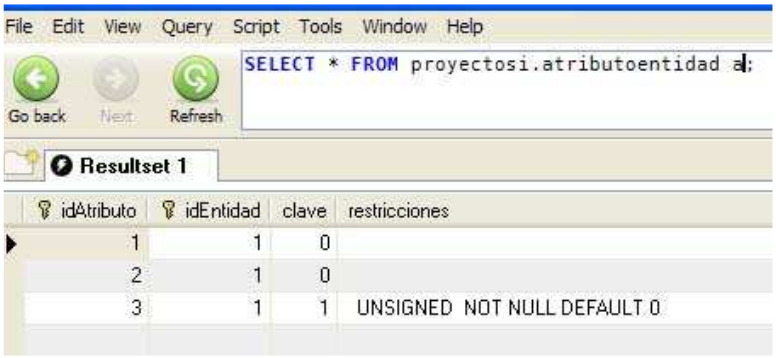
Comprobamos que en el diccionario de datos se han guardado correctamente la información de la entidad, y sus atributos.

Datos de la Entidad:

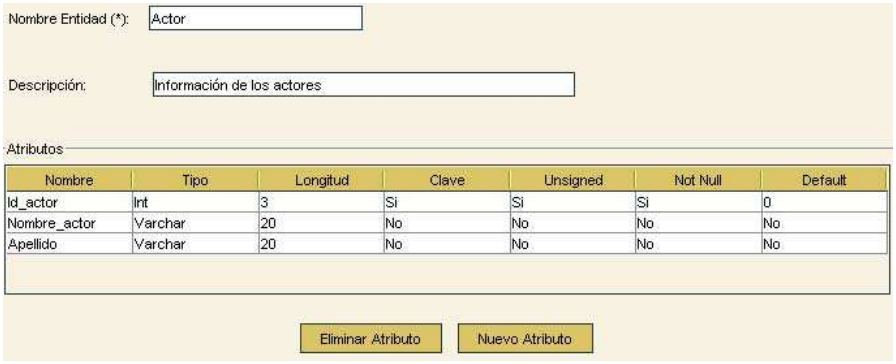


Datos de los atributos:

Datos de los atributos de la entidad:



Creamos con la aplicación las entidades Actor y Estudio.

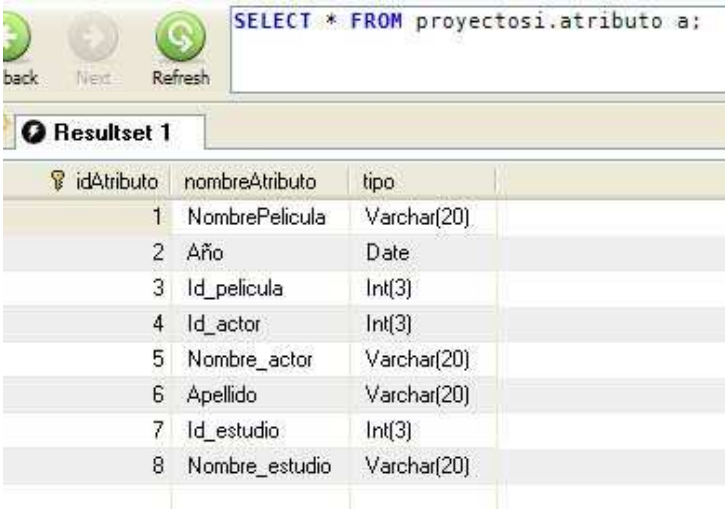


Dichas entidades se crean correctamente.

Comprobamos si se han guardado los datos en el diccionario de datos.

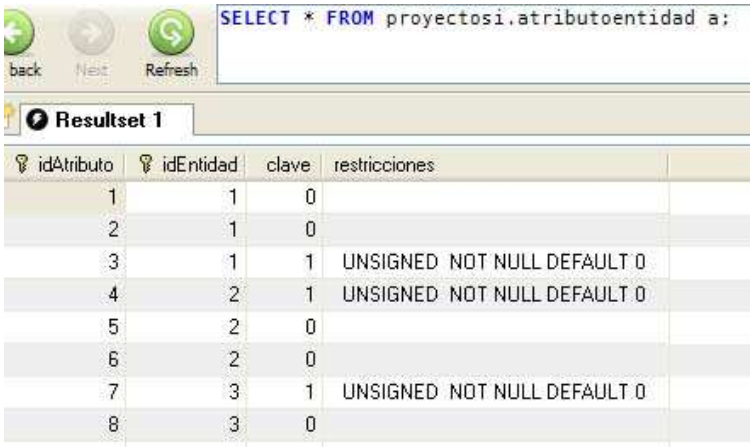
Datos de la Entidad:

Datos de los atributos:



idAtributo	nombreAtributo	tipo
1	NombrePelicula	Varchar(20)
2	Año	Date
3	Id_pelicula	Int(3)
4	Id_actor	Int(3)
5	Nombre_actor	Varchar(20)
6	Apellido	Varchar(20)
7	Id_estudio	Int(3)
8	Nombre_estudio	Varchar(20)

Datos de los atributos de la entidad:

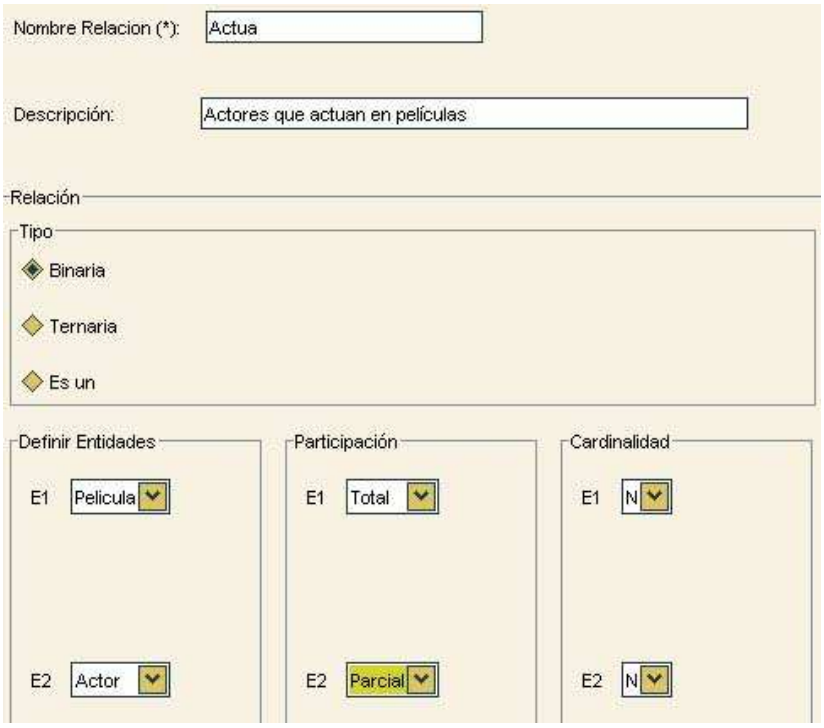


idAtributo	idEntidad	clave	restricciones
1	1	0	
2	1	0	
3	1	1	UNSIGNED NOT NULL DEFAULT 0
4	2	1	UNSIGNED NOT NULL DEFAULT 0
5	2	0	
6	2	0	
7	3	1	UNSIGNED NOT NULL DEFAULT 0
8	3	0	

Observamos como los datos se han guardado correctamente.

Procedemos a crear las relaciones con la aplicación. (Diseño/Relación)

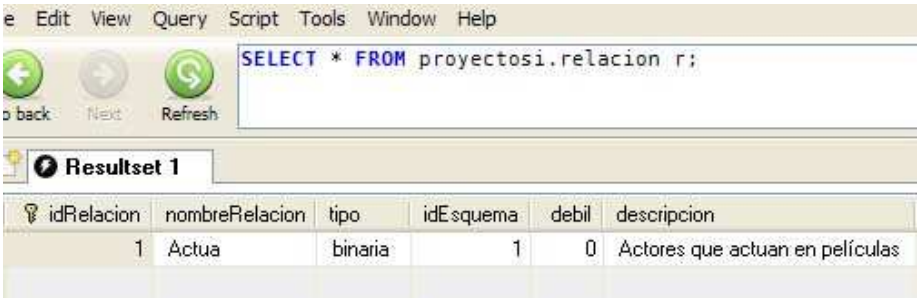
Relación Actúa:



La relación se crea correctamente, y la interfaz principal se actualiza correctamente.

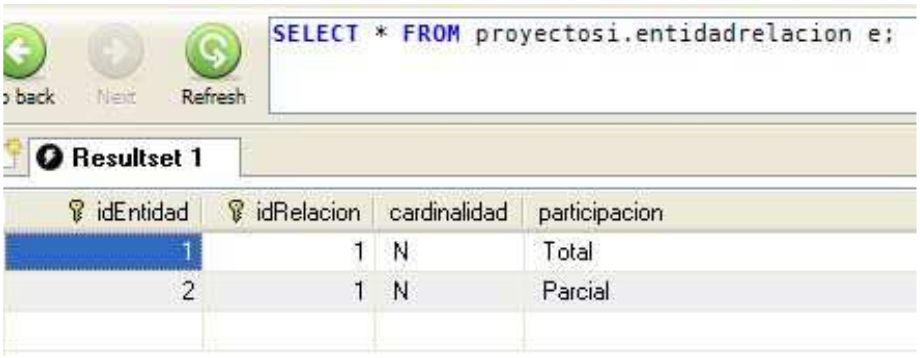
Comprobamos sus datos en el diccionario de datos:

Datos Relación:



idRelacion	nombreRelacion	tipo	idEsquema	debil	descripcion
1	Actua	binaria	1	0	Actores que actuan en películas

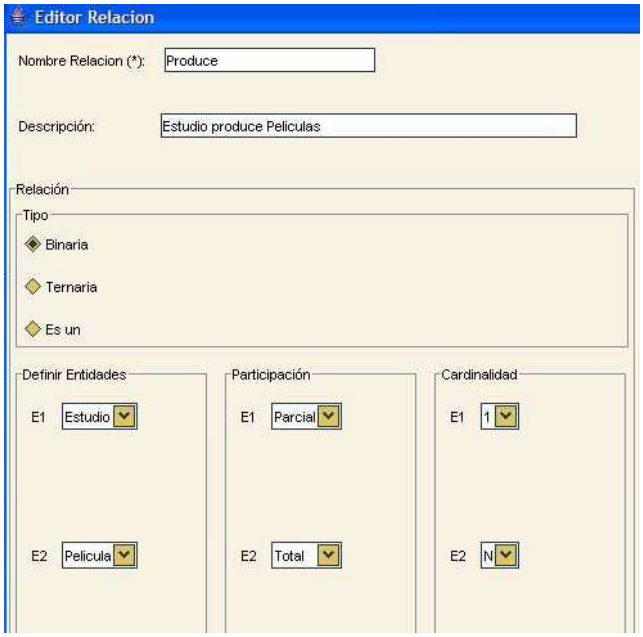
Datos de entidades participantes en la relación:



idEntidad	idRelacion	cardinalidad	participacion
1	1	N	Total
2	1	N	Parcial

La información guardada en el diccionario de datos es correcta.

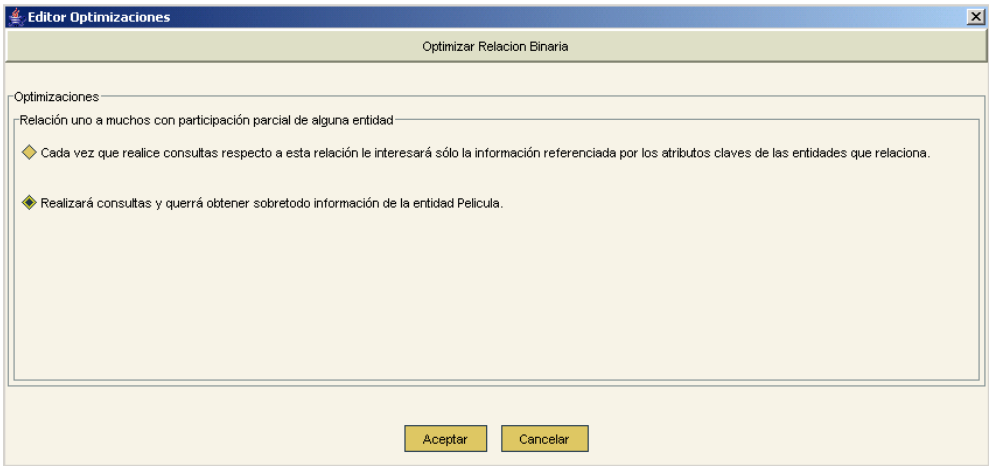
Relación Produce:



Al dar de alta esta relación (relación de uno a muchos con participación parcial de una entidad), se nos pregunta antes si queremos realizar algún tipo de optimización.

En función de la cardinalidad y participación de las entidades, la aplicación nos hace unas preguntas acerca de cómo serán las consultas que queremos realizar con dicha relación. Según la respuesta al tipo de consultas o al tamaño de las entidades consultadas la aplicación aplica la mejor optimización posible.

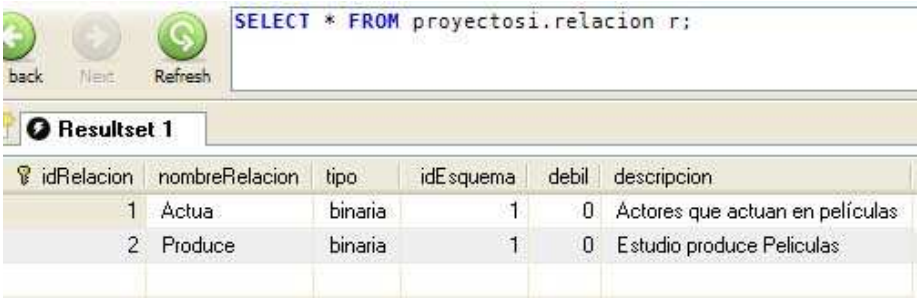
En este caso, seleccionamos "realizará consultas y querrá obtener sobretodo información de la entidad Película".



La relación se crea correctamente.

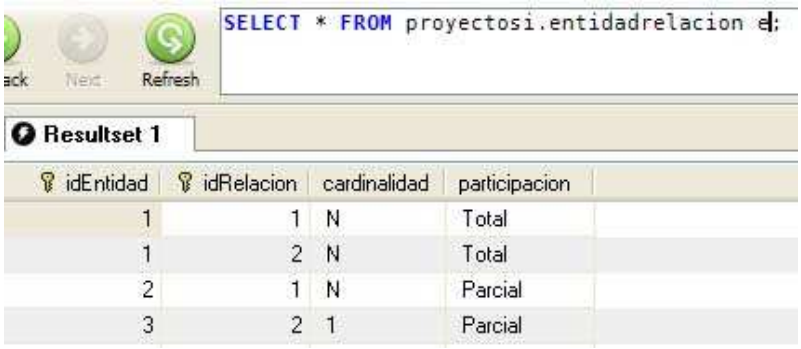
Comprobamos el diccionario de datos:

Datos Relación:



idRelacion	nombreRelacion	tipo	idEsquema	debil	descripcion
1	Actua	binaria	1	0	Actores que actuan en películas
2	Produce	binaria	1	0	Estudio produce Peliculas

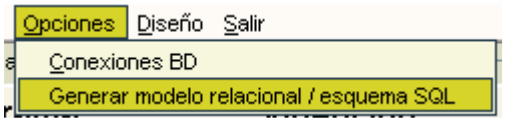
Datos de entidades participantes en la relación:



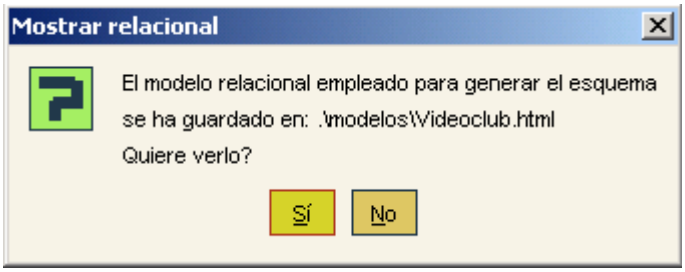
idEntidad	idRelacion	cardinalidad	participacion
1	1	N	Total
1	2	N	Total
2	1	N	Parcial
3	2	1	Parcial

Vemos como los datos de las entidades y las relaciones se han guardado en el diccionario de datos.

Generamos las tablas del Modelo Relacional. (Opciones/Generar modelo relacional / esquema SQL).



Aparece un mensaje indicándonos donde podemos ver el modelo relacional generado así como las sentencias SQL que generan el mismo (éste se encuentra en '.\esquemas\Videoclub.sql').



Si abrimos el archivo vemos cómo el modelo relacional se ajusta al esquema E/R introducido y a las elecciones realizadas.

Modelo Relacional del Esquema: Videoclub

ENTIDADES	ATRIBUTOS
Actor	**idActor** NombreActor Apellidos
Estudio	**idEstudio** Direccion NombreEstudio
Pelicula	año NombrePelicula **Id_pelicula** *idEstudio*
Actua	***idActor Id_pelicula***

Nota:
En **Negrita** se señalan las claves primarias
En *Cursiva* se señalan las claves ajenas

Archivo con código SQL:

```
-- MySQL dump 10.11
--
-- Host: localhost    Database: Videoclub
-- ------------------------------------------------------
-- Server version   5.0.37-community-nt

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `actor`
--

DROP TABLE IF EXISTS `actor`;
CREATE TABLE `actor` (
  `idActor` int(11) NOT NULL,
  `nombre` varchar(20) NOT NULL,
  `apellidos` varchar(50) NOT NULL,
  PRIMARY KEY  (`idActor`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `actor`
--

LOCK TABLES `actor` WRITE;
/*!40000 ALTER TABLE `actor` DISABLE KEYS */;
/*!40000 ALTER TABLE `actor` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `actua`
--
```

```
DROP TABLE IF EXISTS `actua`;
CREATE TABLE `actua` (
  `idActor` int(11) NOT NULL,
  `idPelicula` int(11) NOT NULL,
  PRIMARY KEY  (`idActor`,`idPelicula`),
  KEY `idActor` (`idActor`),
  KEY `idPelicula` (`idPelicula`),
  CONSTRAINT `actua_ibfk_1` FOREIGN KEY (`idActor`) REFERENCES `actor`
(`idActor`),
  CONSTRAINT `actua_ibfk_2` FOREIGN KEY (`idPelicula`) REFERENCES `pelicula`
(`idPelicula`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `actua`
--

LOCK TABLES `actua` WRITE;
/*!40000 ALTER TABLE `actua` DISABLE KEYS */;
/*!40000 ALTER TABLE `actua` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `estudio`
--

DROP TABLE IF EXISTS `estudio`;
CREATE TABLE `estudio` (
  `idEstudio` int(11) NOT NULL,
  `nombreEstudio` varchar(50) NOT NULL,
  `direccion` varchar(50) default NULL,
  PRIMARY KEY  (`idEstudio`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `estudio`
--

LOCK TABLES `estudio` WRITE;
/*!40000 ALTER TABLE `estudio` DISABLE KEYS */;
/*!40000 ALTER TABLE `estudio` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `pelicula`
--

DROP TABLE IF EXISTS `pelicula`;
CREATE TABLE `pelicula` (
  `idPelicula` int(11) NOT NULL,
  `titulo` varchar(30) NOT NULL,
  `año` date default NULL,
  PRIMARY KEY  (`idPelicula`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `pelicula`
--

LOCK TABLES `pelicula` WRITE;
/*!40000 ALTER TABLE `pelicula` DISABLE KEYS */;
/*!40000 ALTER TABLE `pelicula` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `produce`
--
```

```
DROP TABLE IF EXISTS `produce`;
CREATE TABLE `produce` (
  `fecha` date NOT NULL,
  `idEstudio` int(11) NOT NULL,
  `idPelicula` int(11) NOT NULL,
  PRIMARY KEY  (`idPelicula`),
  KEY `idEstudio` (`idEstudio`),
  KEY `idPelicula` (`idPelicula`),
  CONSTRAINT `produce_ibfk_1` FOREIGN KEY (`idEstudio`) REFERENCES `estudio`
(`idEstudio`),
  CONSTRAINT `produce_ibfk_2` FOREIGN KEY (`idPelicula`) REFERENCES `pelicula`
(`idPelicula`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `produce`
--

LOCK TABLES `produce` WRITE;
/*!40000 ALTER TABLE `produce` DISABLE KEYS */;
/*!40000 ALTER TABLE `produce` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2007-06-30 18:50:42
```

Conclusiones:

- La aplicación nos ha permitido modelar de una manera rápida, sencilla e intuitiva la base de datos propuesta como ejemplo.

- La aplicación nos ha permitido crear un nuevo esquema asignándole un usuario y contraseña.

- La aplicación nos ha permitido dar de alta entidades, guardar sus atributos, definir cual es su atributo clave, su tipo, sus restricciones… sin ningún tipo de problemas.

- La aplicación ha funcionado correctamente a la hora de dar de alta relaciones en el sistema. Además hemos comprobado como en función del tipo de relación, participación y cardinalidad el sistema nos ofrece distintas opciones de optimización.

- Hemos comprobado como todos los datos se guardan correctamente en el diccionario de datos.

- Hemos comprobado el correcto funcionamiento del Modelo Vista Controlador, la comunicación entre las capas, el paso de eventos y datos ,.. etc

5. POSIBLES EXTENSIONES

- **Consultas**:
  Permitir que el usuario pueda definir consultas sobre las entidades y relaciones existentes en los diagramas.

- **Datos**:
  Permitir que el usuario pueda poblar las entidades y relaciones con datos.

- **Archivo de Log:**
  Creación de un archivo de Log, en el que se muestren todos los mensajes de error generados por el sistema.

- **Mejoras gráficas en la capa de presentación.**
  Sustituir el actual seleccionador de entidades y relaciones por un árbol (*JTree*) que se situaría en la parte izquierda de la aplicación, y que dado un diagrama mostraría cuales son sus entidades y relaciones.

- **Versión Web multiusuario de la aplicación**.
  Reemplazar la capa de presentación actual, por una capa de presentación Web.
  Para implementar la capa de presentación Web se utilizarían páginas *JSP, el Framework Struts* y Javascript.
  La capa de Lógica y la capa de acceso a datos no tendrían que ser modificadas. Simplemente habría que añadir en estas capas las nuevas funcionalidades necesarias para realizar el control de accesos de múltiples usuarios al sistema.

- **Ampliar a tipos específicos de otros SGBD**
  Si bien esta herramienta puede conectarse a cualquier SGBD cambiando el driver JDBC, sólo podrá realizar cambios a través de ANSI SQL (SQL Estándar). Ampliar a tipos específicos de otros SGBD como Oracle, Postgre SQL, SQL Server permitiría adaptar esta herramienta a SGBD concretos.

6. BIBLIOGRAFÍA

- CEBALLOS, Fco. Javier; *"Java 2: curso de programación"*. Ra-ma. Madrid 2000.

- GAMMA, E., HELM, R., JOHNSON, R., y VLISSIDES, J.; *Patrones de diseño;* Addison Wesley, 2003;

- HOLZNER, Steven; *"La Biblia de Java 2".* Anaya Multimedia, D.L. 2000

- PRESSMAN, Roger. "*Ingeniería del Software Un enfoque Práctico, 5ª Edición*". McGraw Hill.

- SILBERSCHATZ, A., KORTH, H.F., SUDARSHAN, S. ; *Database System Concepts (Fundamentos de bases de datos);* 5ª edición, McGraw-Hill, 2006;

- ULLMAN, J.D.; *Principles of Databases and Knowledge Base Systems;* Computer Science Press, 1998;

- ULLMAN, Larry; *Guía de aprendizaje MySQL*; Pearson Education, S.A. Madrid 2003

- NAVATHE, BATINI, CERI; *Diseño Conceptual de Bases de Datos. Un enfoque Entidad Interrelación;* Addison Wesley, USA, 1992